ISBN 0-7714-2176-1

Automation of the *Aldor* / C++ interface: User Guide

Yannis Chicha, Florence Defaix
and Stephen M. Watt

Technical Report # 537

Department of Computer Science
The University of Western Ontario
London, Canada
N6A 5B7

Automation of the *Aldor*/C++ interface: User Guide

Yannis Chicha, Florence Defaix & Stephen M. Watt
Department of Computer Science
Middlesex College
The University of Western Ontario
London, Ontario, Canada N6A 5B7
{chicha,fdefaix,watt}@csd.uwo.ca

Deliverable 2.2.2

Contents

Chapter 1

Introduction

This user guide provides an overview of the tool designed to facilitate the interoperability between *Aldor* and C++.

One of the principal considerations of the FRISCO project is to provide software tools for the linkage of FRISCO components with technologies currently in use in industrial settings. While much new library development will occur in *Aldor*, the natural inter-operation of *Aldor* and C++ objects becomes a central goal of the project. The language C++ is widely used for application programs in the industrial world and most of the background material of the project (the PoSSo library) is written in C++. Therefore, a well-defined semantic correspondence between the computational models *Aldor* and the C++ is needed in order to provide an interface suitable for direct end use. The first benefit of this work will be to provide for a natural inter-operation of the C++ PoSSo Library and *Aldor* FRISCO Library elements.

This work has been divided into two separate parts. The first one may be called "C++ to *Aldor*" because we have to translate some C++ code in *Aldor* code using the semantics correspondence that has been defined. This application relies on an intermediate XML representation of the C++ source files. So, this part can be divided into two subparts: the generation of XML from C++ and the generation of the C++ and *Aldor* stubs from XML. The implementation of the C++ to XML we made is based on gcc whereas the XML to Aldor part has been written from scratch.

The second part, naturally called "*Aldor* to C++", translates *Aldor* code in C++ code. This comes as a feature of the *Aldor* compiler.

This user guide describes both parts of the interoperability.

Part I

Using C++ from *Aldor*

This part will lead you through the user guide of the "C++ to *Aldor*" translator. The programs supplied enable generating some code in both C++ and *Aldor* languages. It is through this code generation that the interoperability is achieved. It follows the rules defined in [GW97] and summarized in section 4. Knowledge of those basic rules allows the user to make efficient use of this interface.

Here is an example of the need of a well-defined semantics definition between the two languages. Let's say that we want to use the following C++ class in some *Aldor* program:

```
class A {
     int i;
public:
     A(int j): i(j) {}
     void printi() {
          cout << "Here is my value:" << i << endl;
     }
};
```

We have to know that the public part will be accessible in *Aldor* via a Category and that the implementation will be hidden in a Domain. In addition, *Aldor* doesn't hide any argument in the methods declared in the categories, whereas C++ hides *this* which is the object on which the method is applied. That is why it is necessary to add a new argument in the *Aldor* translation of the methods declared in the C++ header file.

The translation of the example above would be:

```
define A__Cat: Category == with {
     printi: % -> ();
}
A: Cat__A with {
     bracket: SingleInteger -> %;  --  the constructor
} == add {
     -- The implementation part looks ugly and
             -- will be decribed later.
}
```

Note: As we will see later in this report, we need to go through the C language to go from C++ to *Aldor* via stub functions.

This implementation relies on an intermediate XML representation of the C++ sources files. The advantage of using this method is to provide a level of abstraction which may be useful for further development, i.e interoperability with languages other than C++

The XML generation from a c++ source is a separate work described in [CDWc98].

Chapter 2

Installation

2.1 Requirements

This section describes what tools are needed to call C++ code from *Aldor*, and how to install the main tool **C++2*Aldor***. This tool generates C++ and *Aldor* stubs from a C++ source code.

C++2*Aldor* uses another tool called **C++2xml** for one of the steps: the generation of an XML representation from a C++ header file. This section deals with the installation of **C++2*Aldor***. For documentation about how to install **C++2xml**, see [CDWc98]. C++ to *Aldor* Version 1.3 requires C++ to XML Version 1.3 which can be found under the name cpp2xml-1.3.tar.gz

2.2 Step by step Installation

1. Check **c++2xml** is installed properly:
   the command `which cpp2xml` should give you the path where the tool is installed. If not, add the corresponding directory to your **PATH** variable.
2. Go into the source directory: `cd c++2aldor-1.3`
3. Edit Makefile and set **INSTALL_DIR** to the directory where you want to install c++2aldor (executable, documentation and examples will be under that directory)
4. Launch the installation: `make`
5. Add **CPP2ALDOR_ROOT** in your environment
   This variable should point to **INSTALL_DIR** and will be used by the application to find scripts and libraries
6. Add **CPP2ALDOR_ROOT/bin** in your **PATH**

No other installation is necessary. You can then go to the `examples` directory in order to test the application.

Chapter 3

Using the Tool

This chapter describes how to use the application and what you need to know in order to use it properly.

We first look at a simple example and how to use the tools with it. We then show how the whole process of the translation is working.

3.1 A Simple example

3.1.1 Presentation

First examine the following C++ class that we want to interface:

```
class employee {
     char*  name;
     short department;
     employee* next;
     static employee* list;
public:
     employee(char* n, int d);
     static void print_list();
     virtual void print() const;
};
```

Let's assume we have the following files:

- `ToBeInterfaced.H` which contains the definition of the above class
- `ToBeInterfaced.C` which is the corresponding implementation for the header file
- `AnotherCode.C` which is a separate file which may be used in the implementation of `ToBeInterfaced.C`
- `MyCodeUsingCpp.as` which will make use of the class and which contains the 'main' function
- `MiscStuff.as` more *Aldor* code ...

3.1.2 How to generate and compile this example

Here are the steps for our example:

1. `g++ -c ToBeInterfaced.C AnotherCode.C`
2. `axiomxl -Fao -Fo MiscStuff.as`
3. Create and edit a file call `excludes.lst` in the current directory Most of the time, it should contain the root directory of standard C and C++ includes. See 3.6 for more explanation.
4. `c++2aldor -d excludes.lst -o Generated ToBeInterfaced.H`
5. `g++ -c -I $CPP2ALDOR_ROOT/lib Generated_cc.cc`
6. `axiomxl -Fao -Fo -Y $CPP2ALDOR_ROOT/lib Generated_as.as`
7. Edit `MyCodeUsingCpp.as` to import `Generated_as.ao`
8. ```
   axiomxl -Fx -C link=g++ -Y $CPP2ALDOR_ROOT/lib MyCodeUsingCpp.as
           ToBeInterfaced.o AnotherCode.o MyCodeUsingCpp.o
           Generated_as.o Generated_cc.o
           $CPP2ALDOR_ROOT/lib/*.o
   ```

3.1.3 What has been generated ?

The generated files are as follow:

Aldor file: "Generated_as.as"

```
#include "axllib.as"
define employee__Cat: Category ==  with {
     print_list: () -> ();
     print: % -> ();
}
employee: employee__Cat with {
     bracket: (String,SingleInteger) -> %;
} == add {
     Rep ==> Record (name : String,department : NONE,next : employee,
                     list : employee);
     import from Rep;
     bracket (var__0: String,var__1: SingleInteger) : %
      == cpp__create__8employeePci0(var__0,var__1);
     print_list () : () == cpp__employee__print__list__8employee0(__obj__);
     print ( __obj__: % ) : () == cpp__employee__print__C8employee0();
}
import {
     -- methods from class employee
     cpp__create__8employeePci0: (String,SingleInteger) -> employee;
     cpp__employee__print__list__8employee0: () -> ();
     cpp__employee__print__C8employee0: employee -> ();
} from Foreign C;
```
```

C++ file: "Generated_cc.C"

```
extern "C" {
     /* methods from class employee */
     employee *cpp_create_8employeePci0(char* var__0,int var__1) {
            return new employee(var__0,var__1);
     }
     void cpp_employee_print_list_8employee0() { employee::print_list(); }
     void cpp_employee_print_C8employee0(employee *__obj__) {
              __obj__->print();
     }
}
```

3.2 Step by step explanations

3.2.1 Compilation of mono-language files

- `g++ -c ToBeInterfaced.C AnotherCode.C`
- `axiomxl -Fao -Fo MiscStuff.as`

These two lines compile additional code that doesn't rely on the interoperability between the two languages.

3.2.2 Generation of C++ and *Aldor* stub files

- `c++2aldor -o Generated ToBeInterfaced.H`

This is the line which does the code generation needed to use C++ classes from *Aldor*. c++2aldor need at least one parameters which is the name of the file to interface. Here, we also specified the basename for the files that are going to be generated. We don't need any of the other **c++2aldor** parameter for this example. (For a description of all available options, see 3.3).

3.2.3 Compilation of the generated files

- `axiomxl -Fao -Fo -Y $CPP2ALDOR_ROOT/lib Generated_as.as`
  This line create an *Aldor* library (.ao) and an object file (.o) from the file Generated_as.as
  The *Aldor* generated file uses special libraries that can be found in $CPP2ALDOR_ROOT/lib/
- `g++ -c -I $CPP2ALDOR_ROOT/lib Generated\_cc.cc`
  This line compile the C++ generated file to an object file (.o). It also uses code from a C++ library we provide.

3.2.4 Final compilation

- edit `MyCodeUsingCpp.as` to import `Generated\_as.ao`
- `axiomxl -Fx -C link=g++ -Y $(CPP2ALDOR_ROOT)/lib MyCodeUsingCpp.as`
  `ToBeInterfaced.o AnotherCode.o MyCodeUsingCpp.o`
  `Generated_as.o Generated_cc.o`
  `$(CPP2ALDOR_ROOT)/lib/*.o`

Then the user aldor program (main program) has to be compiled with all the previously compiled object files. Some points need to be explained:

- The **-C link** option enables the user to choose the linker he wants. In our case we want to link C++ so we use the g++ linker.
- The special libraries still that can be found in $CPP2ALDOR_ROOT/lib/
- The objects files for these libraries must be linked together with all the user and generated file to build the final binary.

3.3 Full syntax

The full syntax of **c++2aldor** is:

```
c++2aldor [-v] [-w] [-s] [-r] [-p prefix] [-d filename] [-m "misc g++ opts"]
          [-o oFilesPrefix] infile
c++2aldor [--verbose] [--extraverbose] [--split] [--noRep] [--prefix prefix]
          [--stdDir filename] [--miscoptions "misc g++ opts"]
          [--outfile oFilesPrefix] infile
c++2aldor -h
c++2aldor --help
```

- **-v** or **–verbose**
  This option will list the options and configuration files. It will also list what is skipped by the code generator with a short explanation.
- **-w** or **–extraverbose**
  In addition to the information displayed by **-v** , this option will list the header files as they are processed.
- **-r** or **–noRep**
  With this option the Aldor representation are not generated. Note that this should be the default as they are not used internally and cannot be exported.
- **-m "misc g++ opts"** or **–misc "misc g++ opts"**
  You can transmit any gcc option needed to compile the source code. It is very useful for extra include directories and flags.
  example: `c++2aldor -m "-I/usr/include/foo -DDEBUG" dummy.H`
- **-o oFilesPrefix** or **–outfile oFilesPrefix**
  This option allows the user to specify the prefix for the C++ and *Aldor* generated files.
  example:
  `c++2aldor -o dummy foo.H` generates dummy_as.as and dummy_cc.C
  whereas `c++2aldor foo.H` generates foo_as.as and foo_cc.C
- **-d** or **–stdDir**
  This option let the user specify the name of the file containing the directories to handle as "standard" directories (see 3.6).
- **-s** or **–split**
  This option is used for large examples whose generated code need to be split so that the *Aldor* can compile it. If you want information on how to use this option, see 3.4
- **-p prefix** or **–prefix prefix**
  This option is usefull if you encounter a *"multiple defined symbol"* error. This can occurs if you have a C++ class with the name of a standard *Aldor* type.
  This option adds a prefix to every type names, top-level variables and function names.
  If you want information on how to use this option, see 7.4

3.4 Split mode

In this section we will first show the steps to generate the same example in split mode. Then we will detail the differences with the **no-split** mode.

3.4.1 How to generate and compile the same example in split mode

1. `g++ -c ToBeInterfaced.C AnotherCode.C`
2. `axiomxl -Fao -Fo MiscStuff.as`
3. `create and edit a file call excludes.lst in the current directory`
4. `c++2aldor -s -d excludes.lst -o Generated ToBeInterfaced.H`
5. `g++ -c -I $CPP2ALDOR_ROOT/lib Generated\_cc.cc`
6. `make -f comp_as_lib`
7. edit `MyCodeUsingCpp.as` to import `Generated\_as.ao`
8. ```
   axiomxl -Fx -C link=g++ -Y $(CPP2ALDOR_ROOT)/lib MyCodeUsingCpp.as
           ToBeInterfaced.o AnotherCode.o MyCodeUsingCpp.o
           gen/*.o Generated_cc.o
           $CPP2ALDOR_ROOT/lib/*.o
   ```

3.4.2 Explanation of the differences

- First, the user must add the -s parameter when calling C++2*Aldor*. This will perform the generation of *Aldor* stubs in several files under the directory **gen**.
- As it would be boring to write a compilation line for each generated *Aldor* file by hand (there can be hundreds of files...), a Makfile, called **comp_as_lib** is automatically generated when the -s option is set. It allows the compilation of *Aldor* stubs to libraries and object files in the **gen** directory.
  (You may want to take a look at comp_as_lib to see the details).
- At least, those libraries and object files must be present in the final compilation directive.

3.4.3 The file xml2as.cfg

When in split mode, **xml2as** generates one *Aldor* file for each C++ header file. This can lead to some compilation errors in at least two cases:

- an include file uses code defined in its parent.
  For example, if we take the following hierarchy:

```
example.C ---- dir1/header1.H ---- dir1/dir2/header11.H
          |                    `- dir1/header12.H
          `- dir1/header2.H
          |
          `- header3.H
```
```

Let's say that `header12.H` uses something defined in `header1.H` before the inclusion. Then `header1.H` and its children should be generated in the same *Aldor* file. If not, the file generated for `header1.H` won't compile.

- Sometimes, the C++ code can include a file containing data for a large enumerated type or array so that it doesn't take too much space in the parent file. For example `header1.H` contains the following code:

```
enum my_codes {
   #include "codes.dat"
   NB_CODES
}
```

and `codes.dat` contains the following code:

```
code1,
code2,
...
code7451,
```

In this example, we can see that `codes.dat` won't compile independently and so it should be generated in the same file as the *Aldor* code corresponding to `header1.H`

The files that are to be grouped (i.e: generated in one file) cannot be found automatically in the current version. But you can list them in a file called **xml2as.cfg**. If such a file exists in the source directory, it will be automatically used by **xml2as**.

Syntax

Each line can contain either:

1. a comment (lines beginning by a '#')
2. a section header (lines beginning by a '@')
   the two sections currently recognized are:

   @blocks All the files listed in this section will be generated in the same file than their direct PARENT.
   @extblocks All the files ending with extensions listed in this section will be generated in the same file than their direct PARENT. You can for example say that files containing non-compiling code end in '.foo' or in '.foo.H'.
3. a file name (in the @blocks section)
4. an extension (in the @extblocks section)

An example

```
#
# xml2as.cfg for the PoSSo example
#
```

```
@blocks
PolyIter.H
PolyRing.H
Poly.H

@extblocks
.inl.H
.inl
```

3.5 Binaries, scripts and variables

Here is the list of the files and variables you will need to use c++2aldor.

CPP2ALDOR_ROOT: This is an environnment variable you must set before using the application. It should point to the installed version.

PATH: You must update your **PATH** variable with the directory **$CPP2ALDOR_ROOT/bin** so that the following applications are available.

bin/xml2as: Generate C++ and Aldor code from XML

bin/c++2aldor: Shell script which calls the c++2xml application and the above program with the right parameters.

misc/Makefile.skeleton: You can use this file as a basis for your own examples. It does the generation and compilation of your application by calling the two above applications and the C++ and *Aldor* compilers.

excludes.lst: The user should use the `-d excludes.lst` option most of the time.

xml2as.cfg: This is an optional file that might be needed sometimes in a generation directory. (see 3.4.3)

3.6 Remarks

1. excludes.lst
   An important point is that `excludes.lst` is a file which should be used with the `-d` option. It should contain the list of the directories where the files from the standard librairies are included. According to this file, `c++2aldor` will generate only short definitions for the types defined in these libraries.

   Here is an example of such a file:

   ```
   /usr/include
   /usr/local/lib-g++
   ```

2. During the generation, we automatically add a #include "axllib.as" statement which is incompatible with the BasicMath library. So you must NOT HAVE "-lbasicmath" in your AXIOMXLARGS variable. Anyway, you should be able to use BasicMath by modifying the generated code (to replace the include statement) and then recompiling it.

3.7 Decomposition of the script "c++2aldor"

The following points give a short explanation of what is performed by the script **c++2aldor**. It is not needed to use the tool but can help to better understand how it works.

1. **Creates a temporary ''dummy.C'' from** `ToBeGenerated.H` in /tmp/c++2aldor-process_number
   This new file contains only one line: `#include "ToBeGenerated.H"`. This step is mandatory because **cpp2xml**, used in the next step works only with `''.C''` files.

2. **Generates an intermediate XML representation from the C++ header file.**
   For that, we use an auxilliary tool, called C++2xml whose syntax is:
   `cpp2xml [-split] [-stdDir excludes.lst] -o /tmp/.../dummy.xml /tmp/.../dummy.C`
   with:
   - excludes.lst a file containing a list of prefix for standard header files. It should appear in all directories where the application is used.
   - with the -split option, the XML representation will contain additional tags, specifying in what file such or such definition was found.
   Rem: when this option is set, XML2Aldor will automatically be in "split mode" (see the following step). This option is set if it was passed to C++2*Aldor*.

   (see C++2xml documentation [CDWc98] for more information on these options)

3. **Generates the *Aldor* and C++ stubs from the XML representation.** The syntax is:

   xml2as `[-p` *prefix*`]` *oPrefix*`_as.as` *oPrefix*`_cc.C toBeInterfaced.xml`

   - the -p *prefix* can be used in case of multiple defined symbols in different libraries. It adds a prefix to every type names, top-level variables and function names.
   - if -split was specified in the previous step, one .as file will be created for each .H in the C++ include hierarchy.

4. **Imports the original file in the generated code.** this means that the line `#include "ToBeGenerated.H"` is added at the top of *oPrefix*`_cc.C`.

Chapter 4

Object Model Correspondence for *Aldor* and C++

This section is essentially a summary of [GW97]. The basic ideas of the correspondence between the object models of *Aldor* and C++ are presented i.e. the foundations on which this implementation is based. It laid out the foundations on which we have designed this tool.

4.1 Summary of the *Aldor* Object Model

The *Aldor* object model is a two-level model based on category and domain. A category is used to specify information about domains. A domain is an environment providing a collection of exported constants like types, functions and variables. A domain can be viewed as an *abstract data type* which defines a distinguished type and a collection of related exports. A category is used to place restrictions on the sort of domains it is prepared to handle and to make a general outline of the domains which it may return. It can be viewed as a *type* for these domains. In a category object, the restrictions and promises are expressed in terms of collections of exports which the domains typed by this category will be required to provide. Conversely, a domain may belong to any number of categories so long as it has the necessary exports.

4.2 Summary of the C++ Object Model

The object model of C++ is based on the class concept. A class is a user-defined data type. Classes in C++ are similar to structures in C, except that the visibility of the class members can be specified explicitly. Class members can be variables and functions. The inheritance model of C++ supports dynamic binding through virtual functions and multiple inheritance. C++ has complicated mechanisms for controlling the visibility of class members. They can be declared as either `public`, `protected` or `private`. Only `public` members are available outside the class. However, if a class `B` inherits items from a class `A`, the protected items of class `A` can be used by the derived class `B`.

4.3 Data Correspondence

Both C++ and *Aldor* have well-defined mechanisms for calling programs written in the C language. The simplest way therefore, to describe the data correspondence between C++ and *Aldor*, is in terms of C data types.

C++ and C The C++ programming language is essentially an upward extension of C, so all the types of C have their well defined meaning in C++, and in practice are represented compatibly. This *de facto* compatibility between C compilers' data layout is exploited by C++ compilers for a straightforward linkage to C programs using `extern "C"`, as in the declaration

```
extern "C" {
        extern char *strcpy(char *);
        extern int   fprintf(FILE *, char *, ...);
}
```

***Aldor* and C** *Aldor*'s first order abstract machine (FOAM) defines a number of types which correspond to types on the target machine (in this case C on top of some operating system). The "`Machine`" package, described in [W+94], exports the types provided by the abstract machine. All *Aldor* values are represented internally as elements of one of these types.

A number of examples of exporting and importing C-defined functions can be found in the directory "`$$AXIOMXLROOT/samples/test`". For instance, the following *Aldor* code is using foreign C functions as first-class values:

```
#include "axllib.as"
#pile

import { sin: BDFlo -> BDFlo;
         cos: BDFlo -> BDFlo;
         tan: BDFlo -> BDFlo } from Foreign(C);

import from DoubleFloat;
import from SingleInteger;

Test(): () == {
      fun(f: BDFlo->BDFlo): () == {
              for x: DoubleFloat in step(11)(0.0, 1.0) repeat {
                      print << x <<"->";
                      print << (f (x::BDFlo))::DoubleFloat;
                      print << newline
              }
      }
      fun(sin);
      fun(cos);
      fun(tan);
}
Test();
```

Types mapping. It is useful to understand how the types will be translated between *Aldor* and C++. The following paragraphs explain the mapping.

Basic types. The type mapping is a tricky issue. Depending on the size of the type, it is passed to C as an immediate value or as a pointer to a value. The mapping between the basic types must therefore be considered carefully.

Some C types have a direct correspondence in Aldor: int, char char *, unsigned int, short int, bool, void and void*.

In *Aldor*, large objects are represented by pointers to the corresponding C basic type which means that *c++Type* * corresponds to *AldorType*. It is the case for long int, long unsigned int, long long int, long long unsigned int, float and double. For these types, we created a library of stub types in C++ and Aldor, named **CppDoubleInteger** and so on. Information on our implementation can be found in the technical part (see [CDWb98]).

In the current version, long double*s* are still not handled.

It is interesting to note that in C++ (or at least in g++), an array of a type is passed as parameter to functions as a pointer to the type. For example `int array[10]` is translated as `int *` by the g++ compiler. Therefore the translation of such a type in *Aldor* is `PrimitiveArray SingleInteger` which maps directly to `int *`.

User types. Two important points are to note carefully. The current version handles only pointer or reference to a type when declared a parameter. They are translated in *Aldor* without any dereferencing. For example `A*` or `A&` are translated in `A`.

Here is an array summarizing the current type translations:

C++ type	*Aldor* type
`int`	`SingleInteger`
`char`	`Character`
`char*`	`String`
`unsigned int`	`Word$Machine`
`short int`	`HalfInteger`
`bool`	`Boolean`
`void`	`()`
`void*`	`Pointer`
`long int`	`CppDoubleInteger`
`long unsigned int`	`CppDoubleWord`
`long long int`	`CppLDoubleInteger`
`long long unsigned int`	`CppLDoubleWord`
`float`	`CppSingleFloat`
`double`	`CppDoubleFloat`
`A*`	`A`
`A&`	`A`
`A**`	`Record A`
`(type)[]`	`PrimitiveArray (translated type)`

In the current version, definitions containing **long double** are skipped.

Unions and Typedefs. All unions and typedefs are translated into `Type`. Any value of type defined by a union or a typedef can this way be passed to and received from the methods calls. However no manipulation can be done on these values. The reason is that unions in C++ and unions in *Aldor* are quite different. We need to define a real mapping between these two types. In the same way no satisfying mapping for typedefs has been found yet. These problems should be fixed in the next version.

Enums are translated in *Aldor* into Enumeration types. For example,

```
enum DAY { monday, tuesday, wednesday, thursday, friday, saturday, sunday}
```

gets translated to

```
DAY == 'monday, tuesday, wednesday, thursday, friday, saturday, sunday';
```

The user has to be aware that it is not possible to give values to enumerable constants in *Aldor*. Thus,

```
enum DAY { monday = 666, tuesday, wednesday, thursday, friday, saturday, sunday}
```

still gets translated to

```
DAY == 'monday, tuesday, wednesday, thursday, friday, saturday, sunday';
```

Therefore, if the user program assumes some specific values for the enumerable constants, it is likely that the application won't work. We will try to fix this problem for the next version.

4.4 Class/Category Correspondence

In C++ classes specify both the sharing of interfaces and the sharing of representation. These aspects are treated separately in *Aldor*, with the specification of type interfaces given by Categories and the specification of type representation given by individual Domains.

All of the exports of a category are derived from the public member functions of the C++ class, but are curried to take an additional first parameter, corresponding to the object from which the C++ method is selected. There is no need to provide exports corresponding to the private member functions.

4.5 Class/Domain Correspondence

To do any computation in *Aldor* it is necessary to specify some concrete *domains*. To finish the model of a class hierarchy, it is necessary to select those classes from which objects will be declared and used. For each of those C++ classes, it will be necessary to build a corresponding *Aldor* domain. This *Aldor* domain will serve as a wrapper, representing its values as pointers to the corresponding C++ objects (if they are extended objects), or as the values themselves (if they are pointers/handles).

Chapter 5

Advanced features

This chapter deals with the generation of *Aldor* and C++ stub code from the tree representation of the C++ header file. We will discuss on the interesting points of the *Aldor* generation i.e the "translation" of types, the C++ generated code and the "extern C" interface.

5.1 *Aldor* generation

Classes are translated into categories and domains. For example, a class `foo` will have the following correspondence in *Aldor* (see 4): the category `foo__Cat` exporting all the public members and the domain `foo` implementing these public members. In the current version only the public members are transcribed, the next release of the program will also transcribe the private members to enable the user to use the implementation of the domain in another domain.

Inheritance is a non-trivial issue. Indeed multiple inheritance is the same in *Aldor* as in C++. With *Aldor* multiple inheritance is only on categories, thus on the exported interface of a type. It is not possible to inherit from several implementations nor from the internal representation of a type. That's why it is needed to use the keyword `default` to implement the inherited methods into the category.

Protection. Currently we handle two types of protection: Private/Protected and Public. The program doesn't handle `protected` separately. The reason is that *Aldor* provides two levels of protection (`private` and `public`) but doesn't supply mechanism for the "hybrid" `protected`.

Data members. There are two kinds of data members: private/protected and public. Both types of data members are part of the representation. Public data members are specifically interfaced using functions to access them: `apply` to read them and `set!` to modify them. `apply` enables the user to access a public data member in a more natural way than a usual function call: `obj.data`. `set!` enables to write code like this one: `obj.data := value`.

Templates are now handled. There is a full section with explanations on that new feature. (See 5.5)

Nested classes are also handled. There is a full section with explanations on that new feature. (See 5.6)

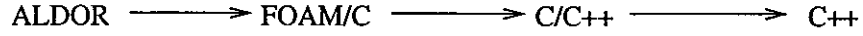
Import. The `import` *Aldor* statement is used to interface *Aldor* and C. In this program it is also used to interface *Aldor* and C++, through C. In a further version of the interoperability between the two languages, the *Aldor* compiler will be modified to accept an `import from Foreign Cplusplus` statement to improve the way the interface can be used.

Library The `#library` statement is used in the generated files to import the definition of the classes and functions in the included files (corresponding to the provided hierarchy).

5.2 The C interface

The option **-GenAldor** is going to generate two files: an *Aldor* source file and a C++ source file. The C++ source file is needed to link *Aldor* to C++. This is done using the C interface provided in *Aldor* (by means of *import* {...} *from Foreign C;*) and in C++ (using *extern "C"* {...}). That's why we provide stub functions called *cpp_....* which are C functions imported in Aldor from C++. These functions are interfaced in a C++ file and call the real C++ methods.

The schema is the following:



The generated file contains a "big" *extern "C"* statement which exports all the stub functions created to interface C and C++. For example if we take the method `void printi();` which is translated in `printi: % -> ();`, the stub function `void cpp__printi(A *obj) { obj->printi();}` will have to be generated.

5.3 Basic types mapping

As explained in the Object Model correspondence chapter, some deferenced basic C++ types have a correspondong Aldor type. For example, **long int *** corresponds to **DoubleInteger**. Using this correspondence, we created a library of stubs types both in C++ and *Aldor* to be able to use the "non deferenced" types (e.g: long int).

Here is the stubs types for **double*** $\Longleftrightarrow$ **DoubleFloat**:

C++ side

```
class CppDoubleFloat {
    double *val;
public:
    CppDoubleFloat(double i);
    ~CppDoubleFloat();
    operator double();
};

extern "C" {
    CppDoubleFloat *createDF(double *d);
    double *convertDF(CppDoubleFloat *d);
}
```

Aldor side

```
CppDoubleFloat: with {
        bracket: DoubleFloat -> %;
        coerce:  % -> DoubleFloat;

} == add {
        bracket(d: DoubleFloat): % == createDF(d);
        coerce(o: %): DoubleFloat == convertDF(o);
}

import {
        createDF: DoubleFloat -> CppDoubleFloat;
        convertDF: CppDoubleFloat -> DoubleFloat;
} from Foreign C;
```

5.4 Static variable accessors

A static variable is a variable whose value doesn't belong to an object's instance but to the class itself. In *Aldor*, the notion of static variable doesn't exist. That's why we had to find a way to get and set a C++ static variable.

1. First, let's see how we use them in C++:
   A C++ class with two static variables

```
class A{
  static int a;
  static int b;
}
```

We need to initialize those variable somewhere in C++:

```
int A::a:=0;
int A::b:=0;
```

And now the way we use it (still in C++):

```
A::a := A::b;
```

or

```
A ex;
ex.a := ex.b;
```

2. Then, look at the way we did *Aldor* accessors for **non-static** variables:

```
define A__Cat: Category ==  with {
   apply: (%,'a')->SingleInteger;
   set!:  (%,'a',SingleInteger)->SingleInteger;
   export from 'a';
   [...]
}
```

Aldor Domain

```
A: A__Cat with {
    bracket: () -> %;
} == add {
   apply(ob: %, f: 'a') : SingleInteger
      == cpp__A__Apply____1A__a(ob);
   set!(ob: %, f: 'a', v: SingleInteger) : SingleInteger
      == cpp__A__Set____1A__a(ob,v);
   ...
}
```

The import from C

```
import {
   -- methods from class A
     cpp__A__Apply____1A__a: A -> SingleInteger;
     cpp__A__Set____1A__a: (A,SingleInteger) -> SingleInteger;
     ...
} from Foreign C;
```

Last, the C code

```
extern ''C'' {
    int cpp_A_Apply__1A_a(A *ob) { return ob->a; }
    int cpp_A_Set__1A_a(A *ob, int v) { return (ob->a = v); }
    ...
}
```

This can be used like that:

```
local ca : A;
ca.a := 3;
```

3. Static variables can also be used in that way. But in order to use them without an instance of the class, we had to add two accessors: a set and and apply with the class name.

```
define A__Cat: Category ==  with {
   apply: (%,'a')->SingleInteger;
   set!:  (%,'a',SingleInteger)->SingleInteger;
   export from 'a';
   apply: ('A','a')->SingleInteger;
   set!:  ('A','a',SingleInteger)->SingleInteger;
   export from 'A';
}

A: A__Cat with {
    bracket: () -> %;
} == add {
   apply(ob: %, f: 'a') : SingleInteger
      == cpp__A__Apply____1A__a(ob);
   set!(ob: %, f: 'a', v: SingleInteger) : SingleInteger
      == cpp__A__Set____1A__a(ob,v);
   apply(c: 'A', f: 'a') : SingleInteger
      == cpp__A__ApplyStatic____1A__a();
   set!(c: 'A', f: 'a', v: SingleInteger) : SingleInteger
      == cpp__A__SetStatic____1A__a(v);
   ...
}
```

The import from C

```
import {
   -- methods from class A
     cpp__A__Apply____1A__a: A -> SingleInteger;
     cpp__A__Set____1A__a: (A,SingleInteger) -> SingleInteger;

     cpp__A__ApplyStatic____1A__a: () -> SingleInteger;
     cpp__A__SetStatic____1A__a: (SingleInteger) -> SingleInteger;
     ...
} from Foreign C;
```

Last, the C code

```
extern ''C'' {
    int cpp_A_Apply__1A_a(A *ob) { return ob->a; }
    int cpp_A_Set__1A_a(A *ob, int v) { return (ob->a = v); }
    int cpp_A_ApplyStatic__1A_a() { return A::a; }
    int cpp_A_SetStatic__1A_a(int v) { return (A::a = v); }
    ...
}
```

We can now use static variables in the same way than in C++ (only we use '.' instead of '::'):
ClassA.variable1
or
a.variable1

5.5 Template handling

5.5.1 Presentation

In C++, template is a way to avoid writing multiple classes or functions with the same behaviour apart from the type used. For example, you might want to create a generic list which can have anything for its cell's content.
In *Aldor*, parameterized types allow the creation of domains that can be used for any type that fits a category. In fact, it is a much more powerful and dynamic concept than C++ templates. So there shouln't be too many difficulties implementing the correspondence. The problem is that the interoperability process uses the C language which doesn't handle templates.
However it is possible to put specialized templates in an extern C. for example :

```
extern "C" {
    foo<A> dummy(foo2<A> *ob){
        return ob->getFoo();
    }
}
```

Moreover, when using a parameterized type on the Aldor side, we need to check that every action made on the type is valid : that is that the type owns at least the set of methods used in the generic class.

5.5.2 Template classes

The solution we chose is to use, for each template parameter, a C++ abstract class which contains any operation made on that type in the template class.
For example, let's take the following template class:

```
// --- Node declaration
template <class T>
class Node {

T *value;
Node<T> *next;
public:
   Node<T> (T *val);
   T *getValue();
   Node<T> *getNext();
   void setNext(Node<T> *n);
};

// --- MyList declaration
template <class T>
class MyList {
public:
   Node<T> *head;
   Node<T> *current;
   Node<T> *last;

   MyList();
   void    addAtEnd(T *val);
   void    rewind();
   Node<T> *getCurrent();
   Node<T> *gotoNext();
   char *toString();
};

template <class T>
MyList<T>::MyList(){
   head    = null;
   current = null;
   last    = null;
}


// --- Node implementation
template <class T>
Node<T>::Node(T *val){
   value = val;
   next = null;
}

template <class T>
T *Node<T>::getValue(){
   return value;
}

template <class T>
Node<T> * Node<T>::getNext(){
   return next;
```

```
}

template <class T>
void Node<T>::setNext(Node<T> *n){
   next = n;
}

// --- MyList implementation
template <class T>
void  MyList<T>::addAtEnd(T *val){
   Node<T> *node = new Node<T>(val);
   if (last == null){
      head = node;
      current = head;
      last = head;
   }
   else{
      last->setNext(node);
      last = last->getNext();
   }
}

template <class T>
void MyList<T>::rewind(){
    current = head;
}

template <class T>
Node<T> *MyList<T>::getCurrent(){
    return current;
}

template <class T>
Node<T> *MyList<T>::gotoNext(){
   if (current != null){
      current = current->getNext();
   }
   return current;
}

template <class T>
char * MyList<T>::toString(){
   char *buf = (char *)
               malloc (1000*sizeof(char));
   char val[20];
   Node<T> *n;

   buf[0] = '\0';
   rewind();
   sprintf(val, "[%s",
     getCurrent()->getValue()->toString());
   strcat(buf,val);

   while ((n = gotoNext())){
      sprintf(val, "; %s",
              n->getValue()->toString());
      strcat(buf,val);
   }
   strcat(buf,"]");
   return buf;
}
```

You can see that there is only one direct call to the T template parameter in MyList's body (toString), and none in Node's body. So these are the abstract classes corresponding to T :

```
class Node_Param_T_Cat{
}

class MyList_Param_T_Cat : Node_Param_T_Cat{
public:
   virtual char * toString();
}
```

Note that MyList_Param_T_Cat must inherits from Node_Param_T_Cat in order to verify the type checking in the generated code.

Important: At the current time those classes are not generated automatically because it would imply to parse the body of the template class. (Remember that we only parse headers) So the user has to write this class and name it with the following scheme:
*TemplateClassName*_Param_*ParamName*_Cat
TemplateClassName is the name of the original class (**MyList** in our example).
ParamName is the name of the parameter for which we are writing the abstract class (**T** in our example).
In the generated code, we assume that this class exists and that it has exactly this name. Moreover, these classes should be written in a file named **'ForTemplates.C'**, included at the beginning of the C++ input file. This is a key name to ensure that the template parameter classes will be compiled first in split mode.

Code generation

1. Generation of a parameterized category for each template class.

```
define Node__Cat(T: Node__Param__T__Cat): Category ==  with {
     getValue: % -> T;
     getNext: % -> Node(T);
     setNext: (%,Node(T)) -> ();
}
```

2. Generation of the corresponding domain. You can note there is an additional parameter to every call to a cpp_xxx function. This is used for *Aldor* prototypes in the import section, as you will see in the corresponding paragraph.

```
Node(T: Node__Param__T__Cat): Node__Cat(T) with {
     bracket: T -> %;
} == add {
     Rep ==> Record (value: T,next: Node(T));
     import from Rep;
     bracket (parm0: T) : % == cpp__create______t4Node1ZX01PX010(T,parm0);
     getValue (ob: %) : T == cpp__Node__getValue____t4Node1ZX010(T,ob);
     getNext (ob: %) : Node(T) == cpp__Node__getNext____t4Node1ZX010(T,ob);
     setNext (ob: %, parm1: Node(T)) : ()
       == cpp__Node__setNext____t4Node1ZX01Pt4Node1ZX010(T,ob,parm1);
}
```

3. Generation of the abstract classes created especially for each parameter. There is nothing special.

```
define Node__Param__T__Cat: Category ==  with {
}

define MyList__Param__T__Cat: Category == Node__Param__T__Cat with {
     toString: % -> String;
}
```

4. Generation of the import section.
   Here we need to have the list of generic parameters with their corresponding categories as first parameters. Indeed, this is the only way to make the type of template parameters known in the import section. And we need to know them for parameters and return types that use template parameters.

```
import {
    -- methods from class Node
    cpp__create______t4Node1ZX01PX010: (T: Node__Param__T__Cat,T) -> Node(T);
    cpp__Node__getValue____t4Node1ZX010: (T: Node__Param__T__Cat,Node(T)) -> T;
    cpp__Node__getNext____t4Node1ZX010: (T: Node__Param__T__Cat,Node(T)) -> Node(T);
    cpp__Node__setNext____t4Node1ZX01Pt4Node1ZX010:
           (T: Node__Param__T__Cat,Node(T), Node(T)) -> ();

    -- methods from class Node_Param_T_Cat
    cpp__Node__Param__T__Cat__toString____16Node__Param__T__Cat0:
            Node__Param__T__Cat -> String;
    -- methods from class MyFloat
    cpp__create______7MyFloatf0: (SingleFloat) -> MyFloat;
    cpp__MyFloat__toString____7MyFloat0: MyFloat -> String;
} from Foreign C;
```

5. Generation of the C code
   First, you can notice the use of instantiated templates in an "Extern "C" " section. Then remember we needed the list of template parameters with their type in the "import from Foreign C " in aldor. As we don't need them in the C code, we just let them as void *.

```
extern "C" {
   /* methods from class Node */
   Node<Node_Param_T_Cat>* cpp_create___t4Node1ZX01PX010(void *parm4,
     Node_Param_T_Cat* parm0) {
            return new Node<Node_Param_T_Cat>(parm0);
   }

   Node_Param_T_Cat* cpp_Node_getValue__t4Node1ZX010(void *parm5,
        Node<Node_Param_T_Cat> *ob) {
            return ob->getValue();
  }

  Node<Node_Param_T_Cat>* cpp_Node_getNext__t4Node1ZX010(void *parm6,
        Node<Node_Param_T_Cat> *ob) {
            return ob->getNext();
  }

  void cpp_Node_setNext__t4Node1ZX01Pt4Node1ZX010(void *parm7,
        Node<Node_Param_T_Cat> *ob, Node<Node_Param_T_Cat>* parm1) {
            ob->setNext(parm1);
  }
}
```

How to use it

To use a C++ template class from Aldor, you need to:

1. Write the abstract classes corresponding to each parameter and their use.
2. Create a C++ class which inherits from these abstract classes.
   For example:

```
class MyFloat : public MyList_Param_T_Cat{
   float i;
  public:
   MyFloat(float j);
   char * toString();
};

MyFloat::MyFloat(float j) : i(j){
}

char * MyFloat::toString(){
  char f[100];
  char *ff;
  sprintf(f,"%f",i);
  ff = strdup(f);
  return ff;
}
```

3. Write your *Aldor* program:

```
#include "t1_as.as"

f():() == {
     import from SingleInteger, SingleFloat, MyFloat;

     local tab2 : PrimitiveArray MyFloat:= new (4,[0.0]);
     tab2.1:=  [1.0];
     tab2.2:=  [2.5];
     tab2.3:=  [3.56788];
     tab2.4:=  [-67.9];
     fl2 : MyList(MyFloat):= [];

     for i in 1..4 repeat {
         addAtEnd(fl2,tab2.i);
         print <<i <<" ---> LIST=" << toString(fl2) << newline;
     }
     print << newline;
}
f()
```

4. Launch the generation and compilation
   Important: Be careful, if you compile it by hand, to include the **".C"** instead of the **".H"** in the generated C code. This is a gcc limitation when using templates.

Template's class with constant parameter

At the current time, such template classes are handled in a special way which should evolve in the future with the Gnu compiler.
A constant template parameter cannot be present in the C stubs because it is a NON instantiated template:

```
template <int n>
void foo() ...
```

So we decided that such classes would be translated to "type" in *Aldor* and so be not directly usable.
Instead, the developer will use stub classes with no constant template parameter. This additional class forwards any method call to the original class with the constant parameter instanciated.
Let's see an example:
If this is the original class.

```
template <class T, int n>
class MyClass{
public:
   Myclass();
   foo<T,n> *doIt();
};
```

Then, one additional class instantiated with the integer 4 is:

```
template <class T>
class MyClass_i4_{
```

```
   MyClass<T,4> *mc;

public:
   Myclass_i4_();
   Foo<T,4> *doIt();
};

template <class T>
MyClass_i4_<T>::MyClass_i4_(){
   mc = new MyClass<T,4>();
}

template <class T>
Foo<T,4> *MyClass_i4_<T>::doIt(){
   return mc->doIt();
}
```

If the user need several instances it is not convenient to write for example ten times such a stub class. In that case, the user can create a couple of macros which create an instance of the class.

```
#define DeclMyclassInst(N)          \
template <class T>                  \
class MyClass_i##N##_{              \
   MyClass<T,##N##> *ml;            \
                                    \
public:                             \
   MyClass_i##N##_();               \
   Foo<T,##N##> *getCurrent();      \
};


#define BodyMyclassInst(N)                      \
template <class T>                              \
MyClass_i##N##_<T>::MyClass_i##N##_(){          \
   mc = new MyClass<T,##N##>();                 \
}                                               \
                                                \
template <class T>                              \
Foo<T,##N##> *MyClass_i##N##_<T>::doIt(){       \
  return mc->doIt();                            \
}
```

In addition to those of "classical" template classes (see 5.5.2), the user needs to follow some steps in order to use a template class with constant parameters:

1. Write the stub class or the macros.
2. Call the macros for each different numbers used in the *Aldor* user program. In our example it can be:
   DeclMyclassInst(76)
   BodyMyclassInst(76)
3. Write the *Aldor*code in that way:

```
local mc76 : MyList__i76__(MyFloat):=[];
doIt(mc76);
```

5.5.3 Template function

This is very similar to template classes. The generation assumes that there is a class called *functionName*_Param_*paramName*_Cat for each template class.

A simple example

First the function definition.

```
template <class T>
void printIt(T *it){
   cout << it->toString()<< endl;
}
```

Here is the abstract class to write for the template parameters 'T'

```
class printIt_Param_T_Cat{
public:
   virtual char * toString(){}
};
```

This is called like that in *Aldor*:

```
printIt(MyFloat)([12.437]);
```

5.5.4 Template function with constant parameter

At the current time template function with constant parameter are not handled partly because we are at the limits of gcc-2.8.0 .

5.6 Nested classes

A nested class (also called inner class) is a class defined in the context of another class or function.

1. First, let's see how we define them in C++:

```
class CC {
public:
    CC(){};

    class BB{       /* BB is an inner class of CC */
    public:
       int aa;
       BB():aa(33){}
       void incr();
    };
};
```

2. Then, how we use it (still in C++):

```
CC::BB bb;
cout << "bb.aa =" << bb.aa << endl;
cout << "increment bb.aa" << endl;
bb.incr();
cout << "bb.aa =" << bb.aa << endl;
cout << "set bb.aa to 5 " << endl;
bb.aa = 5;
cout << "bb.aa =" << bb.aa << endl;
```

3. In Aldor:
   For each inner class we generate first the corresponding category, then another category containing all the constructors. This second category, named `ClassName__D__Cat`, is the type real type of the inner class. The domain is generated inside the domain of the outer class. At last, the import section is generated as usual.
   The generated code for our example gives:

```
#include "axllib.as"
import from Machine, SingleInteger;
define CC__Cat: Category ==  with {
    BB: BB__D__Cat;
}

define BB__Cat: Category ==  with {
   apply: (%,'aa')->SingleInteger;
   set!:  (%,'aa',SingleInteger)->SingleInteger;
   export from 'aa';
   incr: (%) -> ();
}
define BB__D__Cat: Category == BB__Cat with {
   bracket: () -> %;
}
```

```
CC: CC__Cat with {
   bracket: () -> %;
} == add {
   bracket () : % == cpp__create______2CC0();

   BB: BB__D__Cat == add {
      Rep ==> Record (aa: SingleInteger);
      import from Rep;
      apply(ob: %, f: 'aa') : SingleInteger == cpp__BB__Apply__aa(%,ob);
      set!(ob: %, f: 'aa', v: SingleInteger) : SingleInteger == cpp__BB__Set__aa(%,ob,v);
      bracket () : % == cpp__create______Q22CC2BB0(%);
      incr (ob: %) : () == cpp__BB__incr____Q22CC2BB0(%,ob);
   }
}

import {
   -- methods from class CC
   cpp__create______2CC0: ()-> CC;

   -- methods from class BB
   cpp__create______Q22CC2BB0: (T: BB__Cat)-> T;
   cpp__BB__incr____Q22CC2BB0: (T: BB__Cat,BB: T) -> ();
   cpp__BB__Apply__aa: (T: BB__Cat, BB: T) -> SingleInteger;
   cpp__BB__Set__aa: (T :BB__Cat, BB: T,SingleInteger) -> SingleInteger;
} from Foreign C;
```

4. Then, we can use it in Aldor the following way :

```
import from CC, SingleInteger;
bb : BB$CC := [];
print << "bb.aa =" << bb.aa << newline;
print << "increment bb.aa" << newline;
incr(bb)$BB$CC;
print << "bb.aa =" << bb.aa << newline;
print << "set bb.aa to 5 " << newline;
bb.aa :=5;
print << "bb.aa =" << bb.aa << newline;
```

5. Now we can look at the C++ stub generated:

```
extern "C" {
   /* methods from class CC */
   CC* cpp_create___2CC0() { return new CC(); }
   /* methods from class BB */
   CC::BB* cpp_create___Q22CC2BB0(void *dummy) {
       return new CC::BB(); }
   void cpp_BB_incr__Q22CC2BB0(void *dummy,CC::BB *ob) {
       ob->incr(); }
   int cpp_BB_Apply_aa(void *dummy,CC::BB *ob) {return ob->aa;}
   int cpp_BB_Set_aa(void *dummy,CC::BB *ob, int v) {return (ob->aa = v);}
}
```

5.7 Split Aldor generation

The goal is to generate one Aldor Source file by C++ header file and to automate the compilation of these files.

5.7.1 Finding a good solution

Declarations and definitions are put without problem in separated files. If nothing else has been done, difficulties arise at compilation time because each type is known only in the file it has been defined.
Example: let's take the following C++ hierarchy

```
example.C ---- dir1/header1.h ---- dir1/dir2/header11.h
          |                    '- dir1/header12.h
          '-  dir1/header2.h
          |
          '-  header3.h
```

- The first idea was to keep the same hierarchy for Aldor files, to import declarations from the previous level and to export declarations to the next level.

```
examplexC.as ---- dir1/header1xh.as ---- dir1/dir2/header11xh.as
              |                        '- dir1/header12xh.as
              '-  dir1/header2.hxas
              |
              '-  header3.hxas
```

  with the following code in header1xh.as:

```
#library dir1Xdir2Xheader11xh "dir1/dir2/header11xh.ao"
import from dir1Xdir2Xheader11xh;

....

export from dir1Xdir2Xheader11xhl;
```

  In fact everything is not so simple for the following reasons:

  - it is not possible to export from a library in Aldor
  - it is not possible to use libraries with several levels of directories

- So the next try was to flatten the hierarchy and import definitions from the every level under the current level.
  Moreover, Aldor creates a symbol with the twenty-five first characters of every file name, so we had to reverse the order of the directory while the tree is flattened to avoid duplicate symbols with directories like =/usr/local/gcc/lib/gcc-lib/sparc-sun-solaris2.5.1/ .....=

  Our final version can be represented like that:

```
examplexC.as
header1xhXdir1.as
header11xhXdir2Xdir1.as
header12xhXdir1.as
header2xhXdir1.as
header3.hxas
```

with the following code in header1xh.as:

```
#library dir1Xheader121xh "header11xhXdir1.ao"
import from dir1Xdir2Xheader11xh;

#library dir1Xdir2Xheader11xh "dir2Xdir1Xheader11xh.ao"
import from dir1Xdir2Xheader11xh;

...
```

5.7.2 Generating a compilation script

Once the Aldor files have been generated, they must be compiled in the correct order. When there are several files, it can become tricky and very boring to compile them by hand. That's why we added the compilation directives for every generated Aldor file in **comp_as_lib**.

5.8 Operator correspondence

Operators in C++ don't have a one-to-one mapping with operators in Aldor. They have been kept limited for this first version until a mapping for each C++ operators has been decided. However, all the non-translated operators have been kept as normal methods using the g++ internal names. It is less convenient to use but it can still be used ! There are three kinds of operators translations. The **literal** translation translates for example '+' in '+' which is quite reasonable ! Here is the list of operators implied in this kind of translation:

```
+(unary and binary) -(unary and binary) * /
< > <= >=
```

However, it is not possible to do the same for all the operators. For example '=' has not at all the same meaning in C++ and in *Aldor*.

That's why we use a **complete** translation of the operators. Here is the list of translated operators:

```
!= into ~=
== into =
() into apply
[] into apply
```

At last many operators have not been translated yet. They have only been "transliterated". This means that it is possible to use these operators as functions. The names of these functions correspond to the original names except that a _ has been added before special characters and 'C' is added before the name. For example, += is translated as `C_+_=`. This implies that it is not possible to use the infix notation but it is possible to call the function. Here is the list:

```
new delete new[] delete[]
= += -= *= /= % %= && || ++ -- | |= ^ ^= & &= ~ << <<= >> >>=
-> * (unary) & (unary)
?: >? <? ->*
```

The operator '!' has been translated as `C_Not` because of some problem with the *Aldor* compiler.

The operators defined as global functions (typically declared `friend` operators in classes) have not been figured out yet. That's why you will only be able to access them using the internal g++ name. For example the operator << generally overloaded to output something on the screen is to be accessed as `____ls`.

5.9 Compilation error messages (and their solution)

When creating a new C++/Aldor application, it is highly possible that the generated code will not compile at first, especially if the C++ library uses templates. This section presents the most likely compilation error messages and what the user should do in order to remove them.

5.9.1 "... not declared in this scope" or "... undeclared"

(Aldor compiler message)

```
  'Set_Param_T_Cat' was not declared in this scope
or
  'Set_Param_T_Cat' undeclared (first use this function)
```

As explained in section 5.5, for each template class in the C++ header files, the user has to write some classes. In our example, the user should write the following class:

```
class Set_Param_T_Cat{}
```

Note: these classes should be placed in a file '**ForTemplates.C**' included at the beginning of the C++ input file.

5.9.2 "No matching function call for ..."

(C++ compiler message)

```
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.9/installed/include/cmm.h:
  In method 'void mmArray<CmmArray_Param_T_Cat>::traverse()':
  /ptibonum/fdefaix/frisco/PoSSoLib-2.1.9/installed/include/cmm.h:842:
  no matching function for call to 'CmmArray_Param_T_Cat::traverse ()'
```

As explained in section 5.5, when a direct method call is applied to a template parameter, one has to write a corresponding virtual method in the special class.

In our example, the user should write the following in '**ForTemplates.C**':

```
class CmmArray_Param_T_Cat{
public:
   virtual void traverse(){}
}
```

Note: It is important to give an empty body to these functions.

5.9.3 "No meaning for identifier ..."

(Aldor compiler message)
This can come from two distinct issues:

- **group files:**
  As explained in 3.4.3, some header files should explicitly be grouped with their parents in order to compile. For example, if we take the following hierarchy extract:

```
dir1/header1.H ---- dir1/dir2/header11.H
                  '- dir1/header12.H
                  '- dir1/header13.H
```

Let's say that 'dir1/header12.H' and 'dir1/dir2/header11.H' can't be compiled without definitions or includes in 'dir1/header1.H'.
The file '**xml2as.cfg**' should contain the following lines:

```
@blocks
dir1/header12.H
dir1/dir2/header11.H
```

For example, in the PoSSo example IterTest2, the files 'PolyIter.H', 'PolyRing.H' and 'Poly.H' should be grouped with their parent 'Polynomial.H'

```
"./gen/PolyxHXincludeXinstalledXPoSSoLib-2x1x9XfriscoXfdefaixXptibonum.as", line 314:
cpp__PoSSo9CmmObject__traverse____9CmmObject0: (PoSSoCmmObject) -> ();
.....................................................^
[L314 C57] #3 (Error) No meaning for identifier 'PoSSoCmmObject'.
```

The file xml2as.cfg should then contain something like:

```
@blocks
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.9/installed/include/PolyIter.H
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.9/installed/include/PolyRing.H
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.9/installed/include/Poly.H
```

Then the user should launch again the stub generation.

- **#include missing:**
Sometimes C++ libraries are not *perfect*: a header file needed by another one is NOT included in the latter. It will compile if this file has been previously included. There is nothing our tool can do to handle 'historical' definitions.

  For example, in the PoSSo example 'gmp.h' uses _IO_FILE, defined in 'libio.h' without including it. This will lead to the following error message:

```
axiomxl -I. -Y /ptibonum/fdefaix/frisco/inst0125/lib -Y gen -R gen -fo -fao
     gen/gmpxhXincludeXinstalledXPoSSoLib-2x1x9XfriscoXfdefaixXptibonum.as
"./gen/gmpxhXincludeXinstalledXPoSSoLib-2x1x9XfriscoXfdefaixXptibonum.as", line 72:
mpz__inp__binary(parm239: PoSSo____mpz__struct,parm240: PoSSo__IO__FILE):
................^.....................................^
Word$Machine == cpp__PoSSompz__inp__binary0(parm239,parm240);

[L72 C18] #2 (Error) Have determined 1 possible types for the expression.
     Meaning 1: Type, ?
The context requires an expression of type Tuple(Type).
[L72 C18] #3 (Error) Argument 1 of '->' did not match any possible parameter type.
  The rejected type is Type, ?.
  Expected type Tuple(Type).
[L72 C57] #1 (Error) No meaning for identifier 'PoSSo_IO_FILE'.
```

The only solution is to include 'libio.h' in 'gmp.h' before regenerating the stubs.

5.9.4 "undefined reference to ..."

(Linker message)

```
user.as:
IterTest2_cc.o: In function 'cpp_mswCheckAllocatedObj0':
IterTest2_cc.o(.text+0x47c): undefined reference to 'mswCheckAllocatedObj'
#1 (Fatal Error) Linker failed.  Command was: g++ user.o user001.o user002.o
  user003.o user004.o user005.o user006.o user007.o user008.o IterTest2_cc.o
  stubs.o /ptibonum/fdefaix/frisco/inst0125/lib/CppTypes.o
  /ptibonum/fdefaix/frisco/inst0125/lib/CppTypes_as.o axlmain.o
  -L/ptibonum/fdefaix/frisco/inst0125/lib -L.
  -L/ptibonum/fdefaix/frisco/PoSSoLib-2.1.9/installed/lib -L.
  -L/usr/local/aldor1.1.12/share/lib -L/usr/local/aldor1.1.12/lib -o  user
  -lPL_Groebner -lParser -lPL_Gmp -lCpl -laxllib -lfoam
#1 (Warning) Removing file 'user.o'.
#2 (Warning) Removing file 'axlmain.o'.
make: *** [no_split] Error 1
```

This can come from two distinct issues:

- **a library is missing in the command line:**
  The user should check where this function is defined and add the corresponding library in the compilation line (eg: -lfoo if the function is defined in libfoo.a) To check where a function is defined on Unix, the user can use 'nm' and 'grep' as follow:
  `nm --print-file-name *.a | grep 'T ' | grep functionName` If the function is not defined anywhere, see the following solution.

- **a function declared but never defined:**
  C++ linker allow for undefined function when they are not used. However, we generate C/Aldor stub functions even for undefined C++ functions as we have no way to know which version will be used. So if the function is not in any library (see explanation above) the user should create a C file and add the missing functions with an empty body.
  In our example, it would be:

  ```
  void mswCheckAllocatedObj(void *ob) {}
  ```

5.10 How we made the PoSSo Library example work

This section is a step by step explanation on how we made the PoSSo (IterTest2) example. It can be useful to read it before trying to interface a new application.

This explanation worked for us with the system described below and may be different on any other system.

- PC under Linux RedHat 5.2
- gcc-2.8.1, standard compiler for `C` and C++
- cpp2xml v1.3d (on a gcc-2.8.1) / c++2aldor v1.3d
- PoSSo-2.1.10

Whenever a file is modified in a step, we give the new content in step#.fileName All the modified files are presentd at the end of this section.

5.10.1 Step by Step

1. C++ file to handle
   - we chose Posso-2.1.10/lib/Algebra/Poly/IterTest2.C
   - we copied this file in a new directory (called 'working directory' in the following steps) under a new name 'IterTest2b.C'
   - we edited IterTest2b.C to comment the 'main' function (it is written in aldor as the final application)

   $\Longrightarrow$ `[step1.IterTest2b.C]`
2. Creating the Makefile
   - we took Makefile.skeleton from $CPP2ALDOR_ROOT/misc and copied it in the working directory with the name 'Makefile'
   - we set GCC_OPTIONS with the local value
   - we retrieved include files and libraries needed to compile the C++ file.
     `make -n IterTest2 in Posso-2.1.10/lib/Algebra/Poly`
   - we changed the makefile to process IterTest2b.C with those include files and libraries.
     Notes:
     (a) `-L` becomes `-Y` with axiomxl
     (b) ../../.. should be replaced by the home directory of the PoSSo library

   $\Longrightarrow$ `[step2.Makefile]`
3. First generation !
   - `make >& step3.errors`
4. Creating template parameter classes

- Taking a look at the error messages, we noticed some xxxx_Param_T_Cat class missing. As explained in 5.9, one class should be written per template parameter.
  So we created a file named 'ForTemplates.C' in the working directory and added some empty classes.
  We used the following command to find them:
  `grep '_Param_' step3.errors |cut -f 2 -d ' ' |uniq`
  The result is:

```
'CmmArray_Param_T_Cat'
'Set_Param_T_Cat'
'_PL_ListNode_Param_T_Cat'
'PL_List_Param_T_Cat'
'PL_Iterator_Param_T_Cat'
'sl_Param_T_Cat'
```

- We imported this file at the beginning of IterTest2b.C

$\Longrightarrow$ `[step4.ForTemplates.C] [step4.IterTest2.C]`

5. Second generation !
   - `make >& step5.errors`
6. Filling template parameter classes
   - As explained in 5.9, the user has to create empty virtual methods for every method actually called on a template parameter.
     `grep 'no match' step5.errors`

```
.../PoSSoLib-2.1.10/installed/include/cmm.h:842:
   no matching function for call to 'CmmArray_Param_T_Cat::traverse ()'
.../PoSSoLib-2.1.10/installed/include/List.H:536:
   no match for 'PL_List_Param_T_Cat & < PL_List_Param_T_Cat &'
.../PoSSoLib-2.1.10/installed/include/List.H:491:
   no match for '_IO_ostream_withassign & << sl_Param_T_Cat &'
```

   $\Longrightarrow$ `[step6.ForTemplates.C]`
7. Third generation !
   - `make >& step7.errors`
8. Writing the user application
   - we wrote a translation of the IterTest2.C main function in user.as
   - we wanted to use 'cin' and 'cout'. As they are in system headers we have to import them by hand.
   - Note: we modified the Makefile to compile only the final application because we didn't need to regenerate everything

   $\Longrightarrow$ `[step8.user.as] [step8.stubs.C] [step8.Makefile]`
9. C++/*Aldor* Compilation !
   - `make final >& step9.errors`
10. Removing undefined references

- As explained in 5.9, references messages can come from a missing library of a function declared but never defined.
- We first had to list them. In the working directory, we typed the command:
  `grep undefined step9.errors |cut -d ' ' -f5- | uniq`

```
'mswCheckAllocatedObj'
'CmmGetStackBase(void)'
'CmmObject::operator new [](unsigned int, CmmHeap *)'
'CmmObject::operator delete [](void *)'
'mpz_inp_binary'
'mpz_out_binary'
'PL_Coeff::operator int(void)'
'PL_GCoeffRing::outCoeffRing(ostream &) const'
'convertS2B(PL_CoeffRing const &, PL_CoeffRing const &, PL_Coeff const &)'
'PL_Poly::str2poly(char *)'
'PL_PolyRing::isError(PL_Poly const &) const'
'mpq_mul_2exp'
'mpq_add_ui'
'mpq_sub_ui'
'mpq_div_2exp'
```

- Then we had to check if they were defined somewhere in the PoSSo library. In PoSSo-2.1.10/installed/lib, we typed the following:
  `nm --print-file-name -C * |fgrep 'mswCheckAllocatedObj'`
  ...
  `nm --print-file-name -C * |fgrep 'CmmObject::operator delete [](void *)' ...`
- We wrote an empty function/method for each of them because they weren't defined anywhere

$\Longrightarrow$ `[step10.stubs.C]`

11. C++/*Aldor* Compilation !
    - `make final >& step11.errors`
12. Finding the correct Libraries
    - After some tests, we found the correct library line to provide.

    $\Longrightarrow$ `[step12.Makefile]`
13. C++/*Aldor* Compilation !
    - `make final >& step13.errors`
    - the error file was empty, the compilation succeeded.
14. Running the test
    - `user < testI2-2.in`

5.10.2 Modified files in each step

step1.IterTest2b.C

```
//  $Id: IterTest2.C,v 1.8 1998/04/07 15:21:16 passaro Exp $
#ifndef lint
static char vcid[] = "$Id: IterTest2.C,v 1.8 1998/04/07 15:21:16 passaro Exp $";
#endif /* lint */
// File:     IterTest2.C - Polynomial iterators test program -*- C++ -*-
// Package:  PoSSo Polynomial Library.
// Created:  1995 May 12
// Author:   Giovanni A. Cignoni
// E-Mail:   cygnus@di.unipi.it
// Address:  Dipartimento di Informatica
//           Universita' di Pisa
//           Corso Italia 40  56125 Pisa  ITALY
//
// Copyright (C) 1995 Dipartimento di Matematica, POSSO ESPRIT BRP 6846.
//
// This file is part of the POSSO LIBRARY.
//
// This code is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
//
//
#include "GCoeff.H"

// Use GCoeff instead of BCoeff - Giovanni.
#define PL_BCoeffRing            PL_GCoeffRing
#define createBCoeffRing         createGCoeffRing
#include "SZpCoeff.H"
#include "SCoeff.H"
#include "DensePP.H"
#include "TDensePP.H"
#include "Polynomial.H"

void outData( PL_PolyIter& i ) {
  PL_Monomial& m = createMonomial();
  PL_Poly& p     = createPoly();

  m = i.current();
  p = i.poly();

  cout << "  polynomial  = ";
  cout << p;
  cout << ";" << endl;
  cout << "  iterator at = ";
  // cout << m;
  cout << i.current();
  cout << "." << endl;
}

/* main function removed */
```

step2.Makefile

```
#
# This is a skeleton for your Makefiles
#


GCC_OPTIONS=-I/usr/include/g++

#####################################################
POSSO_ROOT=/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10

MORE_INCLUDE = -I. -I$(POSSO_ROOT) -I$(POSSO_ROOT)/installed/include/  -DNDEBUG  -Dlinux=1
 -DPL_CMM=1 -DPL_GENERIC_POLY=1 -DPL_GENERIC_COEFF=1 -DPL_GMP_BIGNUM=1 -DPL_REALSOLVING=1
 -I. -I$(POSSO_ROOT) -I$(POSSO_ROOT)/installed/include/

MORE_LIBS =  -Y$(POSSO_ROOT)/installed/lib/  -Y. -lPoly -lPL_IO -lNFPoly -lGCDPoly -lGroebner
 -lAlgebra  -lRingMaps -lBigInt -lSCoeff -lZpCoeff  -lSZpCoeff -lBigInt -lGCoeff -lFCoeff
 -lHCoeff -lH2Coeff -lF2Coeff  -lDHCoeff -lRCoeff -lProdCoeff -lDPECoeff -lMPCCoeff -lmt -lmpc
 -lInf2Coeff -lErrorCoeff -lCoeff -lPL_Gmp -lPP -lDensePP -lTDensePP -lPDensePP -lBDensePP
 -lHilb -lCmm -lError -lm -lParser -lBigInt -lPL_Gmp
#####################################################

# 1. Choose what kind of generation your want to do as default (split or not)
all: nosplit

# .INIT is not defined in every make ...
# This section checks that you have CPP2ALDOR_ROOT in your environment
# (do not change this section)
MYINIT:
@if test -z "$(CPP2ALDOR_ROOT)"; then \
 echo 'You must have the variable CPP2ALDOR_ROOT set before running make'; \
 false; \
fi

# 2a. Change this section if you want to generate in nosplit mode
nosplit: MYINIT
c++2aldor --noRep -m "$(GCC_OPTIONS) $(MORE_INCLUDE)" IterTest2b.C
g++ -c $(GCC_OPTIONS) $(MORE_INCLUDE) -I $(CPP2ALDOR_ROOT)/lib IterTest2b_cc.C
axiomxl -Fo -Fao  -Y $(CPP2ALDOR_ROOT)/lib IterTest2b_as.as
axiomxl -Fx -C link=g++ -Y $(CPP2ALDOR_ROOT)/lib $(MORE_LIBS) user.as\
  IterTest2b_cc.o IterTest2b_as.o\
  $(CPP2ALDOR_ROOT)/lib/CppTypes.o $(CPP2ALDOR_ROOT)/lib/CppTypes_as.o

# 3. Change this section to fit your application
clean:
\rm -f *.o *.ao user *_cc.C *_as.as *_cc.cc
\rm -rf gen comp_as_lib *~
```

step3.errors

```
c++2aldor --noRep -m "-I/usr/include/g++ -I. -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10
  -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/
  -DNDEBUG  -Dlinux=1 -DPL_CMM=1 -DPL_GENERIC_POLY=1 -DPL_GENERIC_COEFF=1
  -DPL_GMP_BIGNUM=1 -DPL_REALSOLVING=1   -I. -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10
  -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/ " IterTest2b.C
Skip    : __imanip uses function ptr
Skip    : __omanip uses function ptr
Skip    : _IO_free_type uses function ptr
Skip    : __compar_fn_t uses function ptr
Skip    : terminate_handler uses function ptr
Skip    : unexpected_handler uses function ptr
Skip    : new_handler uses function ptr
Skip    : __new_handler uses function ptr
g++ -c -I/usr/include/g++ -I. -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10
  -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/  -DNDEBUG
  -Dlinux=1 -DPL_CMM=1 -DPL_GENERIC_POLY=1 -DPL_GENERIC_COEFF=1 -DPL_GMP_BIGNUM=1
  -DPL_REALSOLVING=1   -I. -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10
  -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/
  -I /ptibonum/fdefaix/frisco/test0601/inst/lib IterTest2b_cc.C
IterTest2b_cc.C:412: 'CmmArray_Param_T_Cat' was not declared in this scope
IterTest2b_cc.C:412: type/value mismatch at argument 0 in template parameter list for
 'template <class T> CmmArray<T>'
IterTest2b_cc.C:412:   expected a type, got '{error}'
IterTest2b_cc.C: In function 'void cpp_t8CmmArray1ZX01_traverse__t8CmmArray1ZX010(void *, int *)':
IterTest2b_cc.C:413: request for member 'traverse' in '*ob', which is of non-aggregate type 'int'
IterTest2b_cc.C: At top level:
IterTest2b_cc.C:421: 'Set_Param_T_Cat' was not declared in this scope
IterTest2b_cc.C:421: type/value mismatch at argument 0 in template parameter list
for 'template <class T> Set<T>'
IterTest2b_cc.C:421:   expected a type, got '{error}'
IterTest2b_cc.C: In function 'int * cpp_create___t3Set1ZX010(void *)':
IterTest2b_cc.C:422: 'Set_Param_T_Cat' undeclared (first use this function)
IterTest2b_cc.C:422: (Each undeclared identifier is reported only once
IterTest2b_cc.C:422: for each function it appears in.)
IterTest2b_cc.C:422: type/value mismatch at argument 0 in template parameter list
for 'template <class T> Set<T>'
IterTest2b_cc.C:422:   expected a type, got '{error}'
IterTest2b_cc.C: At top level:
IterTest2b_cc.C:423: 'Set_Param_T_Cat' was not declared in this scope
IterTest2b_cc.C:423: type/value mismatch at argument 0 in template parameter list for
     'template <class T> Set<T>'
IterTest2b_cc.C:423:   expected a type, got '{error}'
IterTest2b_cc.C:423: type specifier omitted for parameter
IterTest2b_cc.C:423: parse error before '*'
IterTest2b_cc.C: In function 'void cpp_t3Set1ZX01_insert__t3Set1ZX01PX010(...)':
IterTest2b_cc.C:424: 'ob' undeclared (first use this function)
IterTest2b_cc.C:424: 'parm81' undeclared (first use this function)

....[ lots more error messages ] ....
```

step4.ForTemplates.C

```
class CmmArray_Param_T_Cat{};
class Set_Param_T_Cat{};
class _PL_ListNode_Param_T_Cat{};
class PL_List_Param_T_Cat{};
class PL_Iterator_Param_T_Cat{};
class sl_Param_T_Cat{};
```

step4.IterTest2b.C

```
#include "ForTemplates.C"
//  $Id: IterTest2.C,v 1.8 1998/04/07 15:21:16 passaro Exp $
#ifndef lint
static char vcid[] = "$Id: IterTest2.C,v 1.8 1998/04/07 15:21:16 passaro Exp $";
#endif /* lint */
// File:     IterTest2.C - Polynomial iterators test program -*- C++ -*-
// Package:  PoSSo Polynomial Library.
// Created:  1995 May 12
// Author:   Giovanni A. Cignoni
// E-Mail:   cygnus@di.unipi.it
// Address:  Dipartimento di Informatica
//           Universita' di Pisa Corso Italia 40 56125 Pisa ITALY
//
// Copyright (C) 1995 Dipartimento di Matematica, POSSO ESPRIT BRP 6846.
// This file is part of the POSSO LIBRARY.
// This code is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
//

#include "GCoeff.H"

// Use GCoeff instead of BCoeff - Giovanni.
#define PL_BCoeffRing            PL_GCoeffRing
#define createBCoeffRing         createGCoeffRing
#include "SZpCoeff.H"
#include "SCoeff.H"
#include "DensePP.H"
#include "TDensePP.H"
#include "Polynomial.H"

void outData( PL_PolyIter& i ) {
  PL_Monomial& m = createMonomial();
  PL_Poly& p     = createPoly();
  m = i.current();
  p = i.poly();
  cout << "  polynomial  = ";
  cout << p;
  cout << ";" << endl;
  cout << "  iterator at = ";
  cout << i.current();
  cout << "." << endl;
}
```

step5.errors

```
c++2aldor --noRep -m "-I/usr/include/g++ -I. -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10
-I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/ -DNDEBUG -Dlinux=1 -DPL_CMM=1
-DPL_GENERIC_POLY=1 -DPL_GENERIC_COEFF=1 -DPL_GMP_BIGNUM=1 -DPL_REALSOLVING=1   -I.
-I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10
-I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/ " IterTest2b.C
Skip    : __imanip uses function ptr
Skip    : __omanip uses function ptr
Skip    : _IO_free_type uses function ptr
Skip    : __compar_fn_t uses function ptr
Skip    : terminate_handler uses function ptr
Skip    : unexpected_handler uses function ptr
Skip    : new_handler uses function ptr
Skip    : __new_handler uses function ptr
g++ -c -I/usr/include/g++ -I. -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10
 -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/  -DNDEBUG  -Dlinux=1 -DPL_CMM=1
 -DPL_GENERIC_POLY=1 -DPL_GENERIC_COEFF=1 -DPL_GMP_BIGNUM=1 -DPL_REALSOLVING=1   -I.
 -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10 -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/
 -I /ptibonum/fdefaix/frisco/test0601/inst/lib  IterTest2b_cc.C
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/cmm.h:
  In method 'void CmmArray<CmmArray_Param_T_Cat>::traverse()':
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/cmm.h:842:
  no matching function for call to 'CmmArray_Param_T_Cat::traverse ()'
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/List.H:
  In method 'void PL_List<PL_List_Param_T_Cat>::sort()':
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/List.H:536:
  no match for 'PL_List_Param_T_Cat & < PL_List_Param_T_Cat &'
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/List.H:
  In function 'void sl<sl_Param_T_Cat>(class PL_List<sl_Param_T_Cat> &)':
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/List.H:491:
  no match for '_IO_ostream_withassign & << sl_Param_T_Cat &'
/usr/include/g++/iostream.h:108: candidates are: ostream::operator <<(streambuf *)
/usr/include/g++/iostream.h:107:                 ostream::operator <<(ios & (*)(ios &))
/usr/include/g++/iostream.h:106:                 ostream::operator <<(ostream & (*)(ostream &))
/usr/include/g++/iostream.h:102:                 ostream::operator <<(long double)
/usr/include/g++/iostream.h:100:                 ostream::operator <<(float)
/usr/include/g++/iostream.h:99:                 ostream::operator <<(double)
/usr/include/g++/iostream.h:97:                 ostream::operator <<(bool)
/usr/include/g++/iostream.h:95:                 ostream::operator <<(short unsigned int)
/usr/include/g++/iostream.h:94:                 ostream::operator <<(short int)
/usr/include/g++/iostream.h:92:                 ostream::operator <<(long long unsigned int)
/usr/include/g++/iostream.h:91:                 ostream::operator <<(long long int)
/usr/include/g++/iostream.h:89:                 ostream::operator <<(long unsigned int)
/usr/include/g++/iostream.h:88:                 ostream::operator <<(long int)
/usr/include/g++/iostream.h:87:                 ostream::operator <<(unsigned int)
/usr/include/g++/iostream.h:86:                 ostream::operator <<(int)
/usr/include/g++/iostream.h:85:                 ostream::operator <<(const void *)
/usr/include/g++/iostream.h:84:                 ostream::operator <<(const signed char *)
/usr/include/g++/iostream.h:82:                 ostream::operator <<(const unsigned char *)
/usr/include/g++/iostream.h:80:                 ostream::operator <<(const char *)
/usr/include/g++/iostream.h:79:                 ostream::operator <<(signed char)
/usr/include/g++/iostream.h:78:                 ostream::operator <<(unsigned char)
/usr/include/g++/iostream.h:77:                 ostream::operator <<(char)
/.../PoSSoLib-2.1.10/installed/include/List.H:654:  operator <<(ostream &, PL_listable &)
/.../PoSSoLib-2.1.10/installed/include/Coeff.H:190:  operator <<(ostream &, const PL_Coeff &)
```

```
/.../PoSSoLib-2.1.10/installed/include/PP.inl.H:240: operator <<(ostream &, const PL_PP &)
/.../PoSSoLib-2.1.10/installed/include/PP.inl.H:37:  operator <<(ostream &, const PL_PPMonoid &)
/.../PoSSoLib-2.1.10/installed/include/MonomialGPR.inl.H:93: operator <<(ostream &, const PL_Monomial &)
/.../PoSSoLib-2.1.10/installed/include/PolyGPR.inl.H:113:   operator <<(ostream &, PL_Poly &)
/.../PoSSoLib-2.1.10/installed/include/List.H:83:
   operator <<<PL_List_Param_T_Cat>(ostream &, PL_List<PL_List_Param_T_Cat> &)
/.../PoSSoLib-2.1.10/installed/include/List.H:83:
  operator <<<PL_Iterator_Param_T_Cat>(ostream &, PL_List<PL_Iterator_Param_T_Cat> &)
/.../PoSSoLib-2.1.10/installed/include/List.H:83:
  operator <<<sl_Param_T_Cat>(ostream &, PL_List<sl_Param_T_Cat> &)
make: *** [nosplit] Error 1
```

step6.ForTemplates.C

```
class CmmArray_Param_T_Cat{
public:
  void  CmmArray_Param_T_Cat::traverse (){}
};

class Set_Param_T_Cat{};
class _PL_ListNode_Param_T_Cat{};

class PL_List_Param_T_Cat{
public:
  virtual bool& operator< (PL_List_Param_T_Cat &a) {}
};

class PL_Iterator_Param_T_Cat{};

class _IO_ostream_withassign;
class ostream;

class sl_Param_T_Cat{};
_IO_ostream_withassign &  operator<< (_IO_ostream_withassign &b, sl_Param_T_Cat &a) {}
```

step7.errors

```
c++2aldor --noRep -m "-I/usr/include/g++ -I. -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10
-I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/  -DNDEBUG  -Dlinux=1 -DPL_CMM=1
 -DPL_GENERIC_POLY=1 -DPL_GENERIC_COEFF=1 -DPL_GMP_BIGNUM=1 -DPL_REALSOLVING=1 -I.
 -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10 -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/"
 IterTest2b.C
Skip    : __imanip uses function ptr
Skip    : __omanip uses function ptr
Skip    : _IO_free_type uses function ptr
Skip    : __compar_fn_t uses function ptr
Skip    : terminate_handler uses function ptr
Skip    : unexpected_handler uses function ptr
Skip    : new_handler uses function ptr
Skip    : __new_handler uses function ptr
g++ -c -I/usr/include/g++ -I. -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10
 -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/  -DNDEBUG  -Dlinux=1 -DPL_CMM=1
 -DPL_GENERIC_POLY=1 -DPL_GENERIC_COEFF=1 -DPL_GMP_BIGNUM=1 -DPL_REALSOLVING=1   -I.
 -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10 -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include/
 -I /ptibonum/fdefaix/frisco/test0601/inst/lib IterTest2b_cc.C
axiomxl -Fo -Fao  -Y /ptibonum/fdefaix/frisco/test0601/inst/lib IterTest2b_as.as
axiomxl -Fx -C link=g++ -Y /ptibonum/fdefaix/frisco/test0601/inst/lib
 -Y/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib/  -o IterTest2 -Y. IterTest2.o -lPoly -lPL_IO
 -lNFPoly -lGCDPoly -lGroebner -lAlgebra  -lRingMaps -lBigInt -lSCoeff -lZpCoeff -lSZpCoeff -lBigInt
 -lGCoeff -lFCoeff -lHCoeff -lH2Coeff -lF2Coeff -lDHCoeff -lRCoeff -lProdCoeff -lDPECoeff -lMPCCoeff
 -lmt  -lmpc -lInf2Coeff -lErrorCoeff -lCoeff -lPL_Gmp -lPP -lDensePP -lTDensePP -lPDensePP -lBDensePP
 -lHilb -lCmm -lError -lm -lParser -lBigInt -lPL_Gmp user.as\
  IterTest2b.o IterTest2b_cc.o IterTest2b_as.o\
  /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes.o
  /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes_as.o
axiomxl -Fx -C link=g++ -Y /ptibonum/fdefaix/frisco/test0601/inst/lib -Y/ptibonum/fdefaix/frisco/
PoSSoLib-2.1.10/installed/lib/  -Y. -lPoly -lPL_IO -lNFPoly -lGCDPoly -lGroebner -lAlgebra  -lRin
gMaps -lBigInt -lSCoeff -lZpCoeff -lSZpCoeff -lBigInt -lGCoeff -lFCoeff -lHCoeff -lH2Coeff -lF2Co
eff -lDHCoeff -lRCoeff -lProdCoeff -lDPECoeff -lMPCCoeff -lmt  -lmpc -lInf2Coeff -lErrorCoeff -lC
oeff -lPL_Gmp -lPP -lDensePP -lTDensePP -lPDensePP -lBDensePP -lHilb -lCmm -lError -lm -lParser -
lBigInt -lPL_Gmp user.as\
  IterTest2b.o IterTest2b_cc.o IterTest2b_as.o\
  /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes.o /ptibonum/fdefaix/frisco/test0601/inst/li
b/CppTypes_as.o
#1 (Warning) Could not use archive file 'libPL_Gmp.al'.
#2 (Warning) Could not use archive file 'libBigInt.al'.
#3 (Warning) Could not use archive file 'libParser.al'.
#4 (Warning) Could not use archive file 'libm.al'.
#5 (Warning) Could not use archive file 'libError.al'.
#6 (Warning) Could not use archive file 'libCmm.al'.
#7 (Warning) Could not use archive file 'libHilb.al'.
#8 (Warning) Could not use archive file 'libBDensePP.al'.
#9 (Warning) Could not use archive file 'libPDensePP.al'.
#10 (Warning) Could not use archive file 'libTDensePP.al'.
#11 (Warning) Could not use archive file 'libDensePP.al'.
....
#38 (Warning) Could not use archive file 'libNFPoly.al'.
#39 (Warning) Could not use archive file 'libPL_IO.al'.
#40 (Warning) Could not use archive file 'libPoly.al'.
#41 (Fatal Error) Could not open file 'user.as'.
make: *** [final] Error 1
```

step8.Makefile

```
#
# This is a skeleton for your Makefiles
#


GCC_OPTIONS=-I/usr/include/g++

##################################################
POSSO_ROOT=/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10

MORE_INCLUDE = -I. -I$(POSSO_ROOT) -I$(POSSO_ROOT)/installed/include/  -DNDEBUG  -Dlinux=1
 -DPL_CMM=1 -DPL_GENERIC_POLY=1 -DPL_GENERIC_COEFF=1 -DPL_GMP_BIGNUM=1 -DPL_REALSOLVING=1
   -I. -I$(POSSO_ROOT) -I$(POSSO_ROOT)/installed/include/

MORE_LIBS =  -Y$(POSSO_ROOT)/installed/lib/  -Y. -lPoly -lPL_IO -lNFPoly -lGCDPoly -lGroebner
 -lAlgebra  -lRingMaps -lBigInt -lSCoeff -lZpCoeff -lSZpCoeff -lBigInt -lGCoeff -lFCoeff
 -lHCoeff -lH2Coeff -lF2Coeff -lDHCoeff -lRCoeff -lProdCoeff -lDPECoeff -lMPCCoeff -lmt
  -lmpc -lInf2Coeff -lErrorCoeff -lCoeff -lPL_Gmp -lPP -lDensePP -lTDensePP -lPDensePP
-lBDensePP -lHilb -lCmm -lError -lm -lParser -lBigInt -lPL_Gmp
##################################################

# 1. Choose what kind of generation your want to do as default (split or not)
all: nosplit


# .INIT is not defined in every make ...
# This section checks that you have CPP2ALDOR_ROOT in your environment
# (do not change this section)
MYINIT:
@if test -z "$(CPP2ALDOR_ROOT)"; then \
 echo 'You must have the variable CPP2ALDOR_ROOT set before running make'; \
 false; \
fi

# 2a. Change this section if you want to generate in nosplit mode
nosplit: MYINIT
c++2aldor --noRep -m "$(GCC_OPTIONS) $(MORE_INCLUDE)" IterTest2b.C
g++ -c $(GCC_OPTIONS) $(MORE_INCLUDE) -I $(CPP2ALDOR_ROOT)/lib IterTest2b_cc.C
axiomxl -Fo -Fao  -Y $(CPP2ALDOR_ROOT)/lib IterTest2b_as.as

final:
g++ -c $(GXX_INCLUDES) $(MORE_INCLUDES) -I$(POSSO_ROOT)/installed/include stubs.C
axiomxl -Fx -C link=g++ -Y $(CPP2ALDOR_ROOT)/lib $(MORE_LIBS) user.as\
  IterTest2b_cc.o IterTest2b_as.o stubs.o\
  $(CPP2ALDOR_ROOT)/lib/CppTypes.o $(CPP2ALDOR_ROOT)/lib/CppTypes_as.o


# 3. Change this section to fit your application
clean:
\rm -f *.o *.ao user *_cc.C *_as.as *_cc.cc
\rm -rf gen comp_as_lib *~
```

step8.stubs.C

```
#include <iostream.h>

extern "C" {
    istream& cpp_apply_cin() { return cin; }
    ostream& cpp_apply_cout() { return cout; }
}
```

step8.user.as

```
#include "axllib.as"

#include "IterTest2b_as.as";

SI ==> SingleInteger;

import {
cpp__apply__cin: () -> istream;
cpp__apply__cout: () -> ostream;
} from Foreign C;

apply(g: 'Global',c: 'cin'): istream == cpp__apply__cin();
apply(g: 'Global',c: 'cout'): ostream == cpp__apply__cout();


itertest(): () == {
  import from SI, 'Global', 'cin', 'cout';

  local nv: PrimitiveArray(SI) := new (1,0);
  local nd: PrimitiveArray(SI) := new (1,0);

  local buff: PrimitiveArray(SI) := new(1024,0);

  -- Input PPMonoid specifications.

  print << newline;
  print << "Input PPMonoid specifications ..." << newline;

  if (readdirs(Global.cin, nv, nd, buff) = -1) then {

    print << "PPMonoid error." << newline;
    return;
  }


  -- Create a CoeffRing and a PPMonoid.

  cr: PL__CoeffRing := createGCoeffRing();
  ppm: PL__PPMonoid  := createTDensePPMonoid(nv.1, nd.1, buff);

  print << "Using PPMonoid:" << newline;
```

```
    Global.cout << ppm;

    -- Create the PolyRing.

    pr: PL__PolyRing := createPolyRing(cr, ppm);

    setPolyRing(pr);

    m: PL__Monomial := createMonomial();

    p1: PL__Poly := createPoly();
    p2: PL__Poly := createPoly();


    ++ Input test data.

    print << newline;
    print << "Input a monomial (using BCoeff and TDensePP in xyzt) ..." << newline;
    Global.cin >> m;
    print << "Read m = ";
    Global.cout << m;
    print << "." << newline;

    print << newline;
    print << "Input a polynomial (using BCoeff and TDensePP in xyzt) ..." << newline;
    Global.cin >> p1;
    print << "Read p1 = ";
    Global.cout << p1;
    print << "." << newline;

    ...

    ptmp: PL__Poly := createPoly(p1);
    itmp: PL__PolyIter := createPolyIter(ptmp);

    -- Go with tests ...

    print << newline;
    print << "Reset the iterator:" << newline;
    C_=(ptmp,p1);
    reset(itmp,ptmp);
    outData(itmp);

    print << newline;
    print << "Insert monomial m = ";
    Global.cout << m;
    print << ":" << newline;
    C_+_=(itmp,m);
    outData(itmp);

    ...
}
itertest()
```

step9.errors

```
g++ -c   -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include stubs.C
axiomxl -Fx -C link=g++ -Y /ptibonum/fdefaix/frisco/test0601/inst/lib
 -Y/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib/  -Y. -lPoly -lPL_IO -lNFPoly
 -lGCDPoly -lGroebner -lAlgebra  -lRingMaps -lBigInt -lSCoeff -lZpCoeff -lSZpCoeff
 -lBigInt -lGCoeff -lFCoeff -lHCoeff -lH2Coeff -lF2Coeff -lDHCoeff -lRCoeff -lProdCoeff
 -lDPECoeff -lMPCCoeff -lmt  -lmpc -lInf2Coeff -lErrorCoeff -lCoeff -lPL_Gmp -lPP
 -lDensePP -lTDensePP -lPDensePP -lBDensePP -lHilb -lCmm -lError -lm -lParser -lBigInt
 -lPL_Gmp user.as IterTest2b_cc.o IterTest2b_as.o stubs.o\
  /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes.o
 /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes_as.o
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib//libPL_Gmp.a(gmp_additions.o):
  In function 'mpf_get_2dl':
gmp_additions.o(.text+0x0): multiple definition of 'mpf_get_2dl'
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib//libmpc.a(gmptools.o)(.text+0x284):
  first defined here
ld: Warning: size of symbol 'mpf_get_2dl' changed from 76 to 85 in gmp_additions.o
IterTest2b_cc.o: In function 'cpp_mswCheckAllocatedObj0':
IterTest2b_cc.o(.text+0x424): undefined reference to 'mswCheckAllocatedObj'
IterTest2b_cc.o: In function 'cpp_CmmGetStackBase(void, )':
IterTest2b_cc.o(.text+0x700): undefined reference to 'CmmGetStackBase(void)'
IterTest2b_cc.o: In function 'cpp_9CmmObject___vn__9CmmObjectUiP7CmmHeap1':
IterTest2b_cc.o(.text+0x14c0): undefined reference to 'CmmObject::operator new [](unsigned int, CmmHeap *)'
IterTest2b_cc.o: In function 'cpp_9CmmObject___vn__9CmmObjectUiP7CmmHeap0':
IterTest2b_cc.o(.text+0x14e2): undefined reference to 'CmmObject::operator new [](unsigned int, CmmHeap *)'
IterTest2b_cc.o: In function 'cpp_9CmmObject___vd__9CmmObjectPv0':
IterTest2b_cc.o(.text+0x14fc): undefined reference to 'CmmObject::operator delete [](void *)'
IterTest2b_cc.o: In function 'cpp_mpz_inp_binary0':
IterTest2b_cc.o(.text+0x32e4): undefined reference to 'mpz_inp_binary'
IterTest2b_cc.o: In function 'cpp_mpz_out_binary0':
IterTest2b_cc.o(.text+0x3514): undefined reference to 'mpz_out_binary'
IterTest2b_cc.o: In function 'cpp_8PL_Coeff___opi__8PL_Coeff0':
IterTest2b_cc.o(.text+0x595c): undefined reference to 'PL_Coeff::operator int(void)'
IterTest2b_cc.o: In function 'PL_GCoeffRing::cpp_13PL_GCoeffRing_outCoeffRing(ostream &, ) const':
IterTest2b_cc.o(.text+0x66c0): undefined reference to 'PL_GCoeffRing::outCoeffRing(ostream &) const'
IterTest2b_cc.o: In function 'cpp_convertS2B(PL_CoeffRing const &, PL_CoeffRing const &, PL_Coeff const &, )':
IterTest2b_cc.o(.text+0x75d8): undefined reference to 'convertS2B(PL_CoeffRing const &, PL_CoeffRing const &,
              PL_Coeff const &)'
IterTest2b_cc.o: In function 'PL_Poly::cpp_7PL_Poly_str2poly(char *, )':
IterTest2b_cc.o(.text+0xaf60): undefined reference to 'PL_Poly::str2poly(char *)'
IterTest2b_cc.o: In function 'PL_PolyRing::cpp_11PL_PolyRing_isError(PL_Poly const &, ) const':
IterTest2b_cc.o(.text+0xc8c4): undefined reference to 'PL_PolyRing::isError(PL_Poly const &) const'
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib//libmpc.a(gmptools.o): In function 'mpq_round':
gmptools.o(.text+0x141): undefined reference to 'mpq_mul_2exp'
gmptools.o(.text+0x153): undefined reference to 'mpq_add_ui'
gmptools.o(.text+0x161): undefined reference to 'mpq_sub_ui'
gmptools.o(.text+0x170): undefined reference to 'mpq_div_2exp'
#1 (Warning) Could not use archive file 'libPL_Gmp.al'.
#2 (Warning) Could not use archive file 'libBigInt.al'.
#3 (Warning) Could not use archive file 'libParser.al'.
#4 (Warning) Could not use archive file 'libm.al'.
#5 (Warning) Could not use archive file 'libError.al'.
#6 (Warning) Could not use archive file 'libCmm.al'.
#7 (Warning) Could not use archive file 'libHilb.al'.
#8 (Warning) Could not use archive file 'libBDensePP.al'.
```

```
#9 (Warning) Could not use archive file 'libPDensePP.al'.
#10 (Warning) Could not use archive file 'libTDensePP.al'.
#11 (Warning) Could not use archive file 'libDensePP.al'.
#12 (Warning) Could not use archive file 'libPP.al'.
#13 (Warning) Could not use archive file 'libPL_Gmp.al'.
#14 (Warning) Could not use archive file 'libCoeff.al'.
#15 (Warning) Could not use archive file 'libErrorCoeff.al'.
#16 (Warning) Could not use archive file 'libInf2Coeff.al'.
#17 (Warning) Could not use archive file 'libmpc.al'.
#18 (Warning) Could not use archive file 'libmt.al'.
#19 (Warning) Could not use archive file 'libMPCCoeff.al'.
#20 (Warning) Could not use archive file 'libDPECoeff.al'.
#21 (Warning) Could not use archive file 'libProdCoeff.al'.
#22 (Warning) Could not use archive file 'libRCoeff.al'.
#23 (Warning) Could not use archive file 'libDHCoeff.al'.
#24 (Warning) Could not use archive file 'libF2Coeff.al'.
#25 (Warning) Could not use archive file 'libH2Coeff.al'.
#26 (Warning) Could not use archive file 'libHCoeff.al'.
#27 (Warning) Could not use archive file 'libFCoeff.al'.
#28 (Warning) Could not use archive file 'libGCoeff.al'.
#29 (Warning) Could not use archive file 'libBigInt.al'.
#30 (Warning) Could not use archive file 'libSZpCoeff.al'.
#31 (Warning) Could not use archive file 'libZpCoeff.al'.
#32 (Warning) Could not use archive file 'libSCoeff.al'.
#33 (Warning) Could not use archive file 'libBigInt.al'.
#34 (Warning) Could not use archive file 'libRingMaps.al'.
#35 (Warning) Could not use archive file 'libAlgebra.al'.
#36 (Warning) Could not use archive file 'libGroebner.al'.
#37 (Warning) Could not use archive file 'libGCDPoly.al'.
#38 (Warning) Could not use archive file 'libNFPoly.al'.
#39 (Warning) Could not use archive file 'libPL_IO.al'.
#40 (Warning) Could not use archive file 'libPoly.al'.

user.as:
#1 (Fatal Error) Linker failed.  Command was: g++ user.o user001.o user002.o user003.o user004.o
 user005.o user006.o user007.o user008.o user009.o IterTest2b_cc.o IterTest2b_as.o stubs.o
 /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes.o
 /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes_as.o
axlmain.o -L/ptibonum/fdefaix/frisco/test0601/inst/lib
-L/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib/ -L. -L.
-L/usr/local/aldor1.1.12/share/lib -L/usr/local/aldor1.1.12/lib -o  user -lPoly -lPL_IO -lNFPoly
 -lGCDPoly -lGroebner -lAlgebra -lRingMaps -lBigInt -lSCoeff -lZpCoeff -lSZpCoeff -lBigInt
 -lGCoeff -lFCoeff -lHCoeff -lH2Coeff -lF2Coeff -lDHCoeff -lRCoeff -lProdCoeff -lDPECoeff
 -lMPCCoeff -lmt -lmpc -lInf2Coeff -lErrorCoeff -lCoeff -lPL_Gmp -lPP -lDensePP -lTDensePP
 -lPDensePP -lBDensePP -lHilb -lCmm -lError -lm -lParser -lBigInt -lPL_Gmp -laxllib -lfoam
#1 (Warning) Removing file 'user.o'.
#2 (Warning) Removing file 'axlmain.o'.
make: *** [final] Error 1
```

step10.stubs.C

```
#include <iostream.h>

extern "C" {
    istream& cpp_apply_cin() { return cin; }
    ostream& cpp_apply_cout() { return cout; }
}

// undefined methods
#include <cmm.h>

#include <gmp.h>
#include <Coeff.H>
#include "GCoeff.H"
#include <PP.H>
#include <Poly.H>
#include <PolyRing.H>


void mswCheckAllocatedObj(void *ob) {}
void *CmmGetStackBase() {}
void* CmmObject::operator new[](size_t size, CmmHeap* ch= Cmm::heap) {}
void CmmObject::operator delete[](void *ob) {}
size_t mpz_inp_binary  (mpz_ptr, FILE *) {}
size_t mpz_out_binary  (FILE *, mpz_srcptr) {}
PL_Coeff::operator int() {}
ostream& PL_GCoeffRing::outCoeffRing(ostream &os) const{}
PL_Coeff& convertS2B ( const PL_CoeffRing&  sRing, const PL_CoeffRing&  bRing, const PL_Coeff& sCoeff ) {}
PL_Poly& PL_Poly::str2poly( char* s ) {}
int PL_PolyRing::isError( const PL_Poly& p ) const {}
```

step11.errors

```
g++ -c   -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include stubs.C
axiomxl -Fx -C link=g++ -Y /ptibonum/fdefaix/frisco/test0601/inst/lib
 -Y/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib/  -Y. -lPoly -lPL_IO -lNFPoly -lGCDPoly
 -lGroebner -lAlgebra  -lRingMaps -lBigInt -lSCoeff -lZpCoeff -lSZpCoeff -lBigInt -lGCoeff -lFCoeff
 -lHCoeff -lH2Coeff -lF2Coeff -lDHCoeff -lRCoeff -lProdCoeff -lDPECoeff -lMPCCoeff -lmt  -lmpc
 -lInf2Coeff -lErrorCoeff -lCoeff -lPL_Gmp -lPP -lDensePP -lTDensePP -lPDensePP -lBDensePP -lHilb
 -lCmm -lError -lm -lParser -lBigInt -lPL_Gmp user.as\
  IterTest2b_cc.o IterTest2b_as.o stubs.o\
  /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes.o /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes_as.o
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib//libPL_Gmp.a(gmp_additions.o): In function 'mpf_get_2dl':
gmp_additions.o(.text+0x0): multiple definition of 'mpf_get_2dl'
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib//libmpc.a(gmptools.o)(.text+0x284): first defined here
ld: Warning: size of symbol 'mpf_get_2dl' changed from 76 to 85 in gmp_additions.o
/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib//libmpc.a(gmptools.o): In function 'mpq_round':
gmptools.o(.text+0x141): undefined reference to 'mpq_mul_2exp'
gmptools.o(.text+0x153): undefined reference to 'mpq_add_ui'
gmptools.o(.text+0x161): undefined reference to 'mpq_sub_ui'
gmptools.o(.text+0x170): undefined reference to 'mpq_div_2exp'
#1 (Warning) Could not use archive file 'libPL_Gmp.al'.
#2 (Warning) Could not use archive file 'libBigInt.al'.
#3 (Warning) Could not use archive file 'libParser.al'.
#4 (Warning) Could not use archive file 'libm.al'.
#5 (Warning) Could not use archive file 'libError.al'.
#6 (Warning) Could not use archive file 'libCmm.al'.
#7 (Warning) Could not use archive file 'libHilb.al'.
#8 (Warning) Could not use archive file 'libBDensePP.al'.
#9 (Warning) Could not use archive file 'libPDensePP.al'.
#10 (Warning) Could not use archive file 'libTDensePP.al'.
#11 (Warning) Could not use archive file 'libDensePP.al'.
#12 (Warning) Could not use archive file 'libPP.al'.
#13 (Warning) Could not use archive file 'libPL_Gmp.al'.
#14 (Warning) Could not use archive file 'libCoeff.al'.
#15 (Warning) Could not use archive file 'libErrorCoeff.al'.
#16 (Warning) Could not use archive file 'libInf2Coeff.al'.
#17 (Warning) Could not use archive file 'libmpc.al'.
#18 (Warning) Could not use archive file 'libmt.al'.
#19 (Warning) Could not use archive file 'libMPCCoeff.al'.
#20 (Warning) Could not use archive file 'libDPECoeff.al'.
#21 (Warning) Could not use archive file 'libProdCoeff.al'.
#22 (Warning) Could not use archive file 'libRCoeff.al'.
#23 (Warning) Could not use archive file 'libDHCoeff.al'.
#24 (Warning) Could not use archive file 'libF2Coeff.al'.
#25 (Warning) Could not use archive file 'libH2Coeff.al'.
#26 (Warning) Could not use archive file 'libHCoeff.al'.
#27 (Warning) Could not use archive file 'libFCoeff.al'.
#28 (Warning) Could not use archive file 'libGCoeff.al'.
#29 (Warning) Could not use archive file 'libBigInt.al'.
#30 (Warning) Could not use archive file 'libSZpCoeff.al'.
#31 (Warning) Could not use archive file 'libZpCoeff.al'.
#32 (Warning) Could not use archive file 'libSCoeff.al'.
#33 (Warning) Could not use archive file 'libBigInt.al'.
#34 (Warning) Could not use archive file 'libRingMaps.al'.
#35 (Warning) Could not use archive file 'libAlgebra.al'.
#36 (Warning) Could not use archive file 'libGroebner.al'.
```

```
#37 (Warning) Could not use archive file 'libGCDPoly.al'.
#38 (Warning) Could not use archive file 'libNFPoly.al'.
#39 (Warning) Could not use archive file 'libPL_IO.al'.
#40 (Warning) Could not use archive file 'libPoly.al'.

user.as:
#1 (Fatal Error) Linker failed.  Command was: g++ user.o user001.o user002.o user003.o user004.o
 user005.o user006.o user007.o user008.o user009.o IterTest2b_cc.o IterTest2b_as.o stubs.o
 /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes.o
 /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes_as.o axlmain.o
 -L/ptibonum/fdefaix/frisco/test0601/inst/lib -L/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib/
 -L. -L. -L/usr/local/aldor1.1.12/share/lib -L/usr/local/aldor1.1.12/lib -o  user -lPoly -lPL_IO
 -lNFPoly -lGCDPoly -lGroebner -lAlgebra -lRingMaps -lBigInt -lSCoeff -lZpCoeff -lSZpCoeff -lBigInt
 -lGCoeff -lFCoeff -lHCoeff -lH2Coeff -lF2Coeff -lDHCoeff -lRCoeff -lProdCoeff -lDPECoeff -lMPCCoeff
 -lmt -lmpc -lInf2Coeff -lErrorCoeff -lCoeff -lPL_Gmp -lPP -lDensePP -lTDensePP -lPDensePP -lBDensePP
 -lHilb -lCmm -lError -lm -lParser -lBigInt -lPL_Gmp -laxllib -lfoam
#1 (Warning) Removing file 'user.o'.
#2 (Warning) Removing file 'axlmain.o'.
make: *** [final] Error 1
```

step12.Makefile

```
#
# This is a skeleton for your Makefiles
#


GCC_OPTIONS=-I/usr/include/g++

##################################################
POSSO_ROOT=/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10

MORE_INCLUDE = -I. -I$(POSSO_ROOT) -I$(POSSO_ROOT)/installed/include/  -DNDEBUG  -Dlinux=1 -DPL_CMM=1
 -DPL_GENERIC_POLY=1 -DPL_GENERIC_COEFF=1 -DPL_GMP_BIGNUM=1 -DPL_REALSOLVING=1   -I. -I$(POSSO_ROOT)
 -I$(POSSO_ROOT)/installed/include/

MORE_LIBS= -lPL_IO -lGroebner  -lPL_Gmp -lCpl -lBigInt -lParser -lBigInt -Y.  -Y$(POSSO_ROOT)/installed/lib

##################################################


# 1. Choose what kind of generation your want to do as default (split or not)
all: nosplit


# .INIT is not defined in every make ...
# This section checks that you have CPP2ALDOR_ROOT in your environment
# (do not change this section)
MYINIT:
@if test -z "$(CPP2ALDOR_ROOT)"; then \
 echo 'You must have the variable CPP2ALDOR_ROOT set before running make'; \
 false; \
fi

# 2a. Change this section if you want to generate in nosplit mode
nosplit: MYINIT
c++2aldor --noRep -m "$(GCC_OPTIONS) $(MORE_INCLUDE)" IterTest2b.C
g++ -c $(GCC_OPTIONS) $(MORE_INCLUDE) -I $(CPP2ALDOR_ROOT)/lib IterTest2b_cc.C
axiomxl -Fo -Fao  -Y $(CPP2ALDOR_ROOT)/lib IterTest2b_as.as

final:
g++ -c $(GXX_INCLUDES) $(MORE_INCLUDES) -I$(POSSO_ROOT)/installed/include stubs.C
axiomxl -Fx -C link=g++ -Y $(CPP2ALDOR_ROOT)/lib $(MORE_LIBS) user.as\
  IterTest2b_cc.o IterTest2b_as.o stubs.o\
  $(CPP2ALDOR_ROOT)/lib/CppTypes.o $(CPP2ALDOR_ROOT)/lib/CppTypes_as.o


# 3. Change this section to fit your application
clean:
\rm -f *.o *.ao user *_cc.C *_as.as *_cc.cc
\rm -rf gen comp_as_lib *~
```

step13.errors

```
g++ -c   -I/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/include stubs.C
axiomxl -Fx -C link=g++ -Y /ptibonum/fdefaix/frisco/test0601/inst/lib -lPL_IO -lGroebner
  -lPL_Gmp -lCpl -lBigInt -lParser -lBigInt -Y.
  -Y/ptibonum/fdefaix/frisco/PoSSoLib-2.1.10/installed/lib user.as\
  IterTest2b_cc.o IterTest2b_as.o stubs.o\
  /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes.o
  /ptibonum/fdefaix/frisco/test0601/inst/lib/CppTypes_as.o
#1 (Warning) Could not use archive file 'libBigInt.al'.
#2 (Warning) Could not use archive file 'libParser.al'.
#3 (Warning) Could not use archive file 'libBigInt.al'.
#4 (Warning) Could not use archive file 'libCpl.al'.
#5 (Warning) Could not use archive file 'libPL_Gmp.al'.
#6 (Warning) Could not use archive file 'libGroebner.al'.
#7 (Warning) Could not use archive file 'libPL_IO.al'.

user.as:
```

Chapter 6

Current limitations

Here is a list of the current limitations and known bugs:

Limitation or bug	Comments
Ellipsis not handled	e.g void f(int,...); the function is not generated
Bitfields not handled	they are not generated
Templates with constant parameters	Template classes with const parameters are sir Templated functions with const parameters ar
init values	need to deal with r-values which can be compl
destructors not handled	they are not generated
static modifier in external functions	
type conversion	e.g: assignment of a char[] to a char*
friend modifier	
"Black list" : mod,in,rem,quo,fix,add not handled	they are not generated
type operators (e.g operator int) not handled	in aldor generation : gcc internal name
infix operator with more than 2 parameters	in aldor generation : not infix but transliterat
friend operators	only imported "from Foreign C" using gcc int
nested classes	if in the methods of the nested class contain a place than the first place, the result won't cor it is possible to modify the generated files by l the PoSSo library doesn't seem to have such c

Chapter 7

Examples

All of the following examples can be generated in the two modes: split or not. Explications are given for the no-split version, you can you in the Makefiles to see how the split version work.

7.1 First tiny example: Complex

In this first example, we import a C++ class defining Complex numbers and a function allowing to print them.

7.1.1 C++ header file

```
class Complex {
   int real_part;
   int im_part;
 public:
   Complex(int i, int j);
   int re() { return real_part; }
   int im() { return im_part; }
   virtual void print();
};
```

7.1.2 C++ implementation

```
#include <iostream.h>
#include "example.hh"

Complex::Complex(int i, int j) {
  real_part = i;
  im_part = j;
}
```

```
void Complex::print() {
  cout << "Complex is : " << re() << " + " << im() << "i" << endl;
}
```

7.1.3 Aldor test file

```
#include "axllib.as"
#include "example_as.as"

f(): () == {

        import from Complex, SingleInteger;
        c: Complex := [3,5];
        print << "Aldor speaking ... I call a C++ function" << newline;
        print(c);
        print << "Back to Aldor" << newline;
}

f()
```

7.1.4 Generation step by step

Instead of the following lines you can also type `make`.

```
g++ -c example.cc
c++2aldor example.hh ex
axiomxl -Fx -C link=g++ user.as *.o
```

7.1.5 Output

When launching the program user:

```
Aldor speaking ... I call a C++ function
Complex is : 3 + 5i
Back to Aldor
```

7.1.6 Aldor generated code

```
.....

#include "axllib.as"
import from Machine, SingleInteger;
#library CppTypes "CppTypes__as.ao"
import from CppTypes;
define Complex__Cat: Category ==  with {
        re: (%) -> SingleInteger;
```

```
        im: (%) -> SingleInteger;
        print: (%) -> ();
}
Complex: Complex__Cat with {
        bracket: (SingleInteger,SingleInteger) -> %;
} == add {
        Rep ==> Record (real__part: SingleInteger,im__part: SingleInteger);
        import from Rep;
        bracket (parm0: SingleInteger,parm1: SingleInteger) : % == cpp__create__
____7Complexii0(parm0,parm1);
        re (ob: %) : SingleInteger == cpp__Complex__re____7Complex0(ob);
        im (ob: %) : SingleInteger == cpp__Complex__im____7Complex0(ob);
        print (ob: %) : () == cpp__Complex__print____7Complex0(ob);
}

import {

        -- methods from class Complex
        cpp__create______7Complexii0: (SingleInteger,SingleInteger)-> Complex;
        cpp__Complex__re____7Complex0: (Complex) -> SingleInteger;
        cpp__Complex__im____7Complex0: (Complex) -> SingleInteger;
        cpp__Complex__print____7Complex0: (Complex) -> ();
} from Foreign C;
```

7.1.7 C++ generated code

```
#include "example.h"
#include "CppTypes.H"
extern "C" {
/* methods from class Complex */
        Complex* cpp_create___7Complexii0(int parm0,int parm1) {
                return new Complex(parm0,parm1); }
        int cpp_Complex_re__7Complex0(Complex *ob) {
                return ob->re(); }
        int cpp_Complex_im__7Complex0(Complex *ob) {
                return ob->im(); }
        void cpp_Complex_print__7Complex0(Complex *ob) {
                ob->print(); }
}
```

7.2 Second tiny example: Fancy Complex

7.2.1 C++ header file

```
class Complex {
  int real_part;
  int im_part;
public:
  Complex(int i, int j);
  int re() { return real_part; }
  int im() { return im_part; }
  virtual void print();
};

class FancyComplex: public Complex {
  char *title;
public:
  FancyComplex(char *s, int k, int h);
  char *getTitle() { return title; }
  void print();
};
```

7.2.2 C++ implementation

```
#include <string.h>
#include <iostream.h>
#include "example.hh"

Complex::Complex(int i, int j) {
  real_part = i;
  im_part = j;
}

void Complex::print() {
  cout << "Complex is : " << re() << " + " << im() << "i" << endl;
}

FancyComplex::FancyComplex(char *s, int k, int h): Complex(k,h) {
  title = strdup(s);
}

void FancyComplex::print() {
  cout << "The complex " << title << " is worth: " << endl;
  Complex::print();
```

7.2.3 Aldor test file

```
#include "axllib.as"
#include "example_as.as"

f(): () == {

        import from FancyComplex, SingleInteger, String;
        c: FancyComplex := ["Three-Five",3,5];
        print << "Title !!! : " << getTitle(c) << newline;
        print << "Aldor speaking ... I call a C++ function" << newline;
        print(c);
        print << "Back to Aldor" << newline;
}

f()
```

7.2.4 Generation step by step

Instead of the following lines you can also type make.

```
g++ -c example.cc
c++2aldor example.hh ex
axiomxl -Fx -C link=g++ user.as *.o
```

7.2.5 Output

```
Title !!! : Three-Five
Aldor speaking ... I call a C++ function
The complex Three-Five is worth:
Complex is : 3 + 5i
Back to Aldor
```

7.2.6 Aldor generated code

```
#include "axllib.as"
import from Machine, SingleInteger;
#library CppTypes "CppTypes__as.ao"
import from CppTypes;
define Complex__Cat: Category ==  with {
        re: (%) -> SingleInteger;
        im: (%) -> SingleInteger;
        print: (%) -> ();
}
Complex: Complex__Cat with {
        bracket: (SingleInteger,SingleInteger) -> %;
} == add {
        Rep ==> Record (real__part: SingleInteger,im__part: SingleInteger);
        import from Rep;
        bracket (parm0: SingleInteger,parm1: SingleInteger) : %
        == cpp__create______7Complexii0(parm0,parm1);
        re (ob: %) : SingleInteger == cpp__Complex__re____7Complex0(ob);
        im (ob: %) : SingleInteger == cpp__Complex__im____7Complex0(ob);
        print (ob: %) : () == cpp__Complex__print____7Complex0(ob);
}
define FancyComplex__Cat: Category == Complex__Cat with {
        -- Inheritance from Complex --
        default {
                re (ob: %) : SingleInteger
                 == cpp__Complex__re____7Complex0(ob pretend Complex);
                im (ob: %) : SingleInteger
                 == cpp__Complex__im____7Complex0(ob pretend Complex);
                print (ob: %) : ()
                 == cpp__Complex__print____7Complex0(ob pretend Complex);
        }
        default {
        }
        getTitle: (%) -> String;
}

FancyComplex: FancyComplex__Cat with {
        bracket: (String,SingleInteger,SingleInteger) -> %;
} == add {
        Rep ==> Record (title: String,real__part : SingleInteger,
                        im__part : SingleInteger);
        import from Rep;
```

```
        bracket (parm2: String,parm3: SingleInteger,parm4: SingleInteger) : %
          == cpp__create______12FancyComplexPcii0(parm2,parm3,parm4);
        getTitle (ob: %) : String
          == cpp__FancyComplex__getTitle____12FancyComplex0(ob);
        print (ob: %) : ()
          == cpp__FancyComplex__print____12FancyComplex0(ob);
}

import {

        -- methods from class Complex
        cpp__create______7Complexii0: (SingleInteger,SingleInteger)-> Complex;
        cpp__Complex__re____7Complex0: (Complex) -> SingleInteger;
        cpp__Complex__im____7Complex0: (Complex) -> SingleInteger;
        cpp__Complex__print____7Complex0: (Complex) -> ();

        -- methods from class FancyComplex
        cpp__create______12FancyComplexPcii0:
             (String,SingleInteger, SingleInteger)-> FancyComplex;
        cpp__FancyComplex__getTitle____12FancyComplex0:
             (FancyComplex) -> String;
        cpp__FancyComplex__print____12FancyComplex0: (FancyComplex) -> ();
} from Foreign C;
```

.....

7.2.7 C++ generated code

```
#include "example.h"
#include "CppTypes.H"
extern "C" {
/* methods from class Complex */
        Complex* cpp_create___7Complexii0(int parm0,int parm1) {
                return new Complex(parm0,parm1); }
        int cpp_Complex_re__7Complex0(Complex *ob) {
                return ob->re(); }
        int cpp_Complex_im__7Complex0(Complex *ob) {
                return ob->im(); }
        void cpp_Complex_print__7Complex0(Complex *ob) {
                ob->print(); }
/* methods from class FancyComplex */
        FancyComplex* cpp_create___12FancyComplexPcii0
                        (char* parm2,int parm3,int parm4) {
                return new FancyComplex(parm2,parm3,parm4); }
        char* cpp_FancyComplex_getTitle__12FancyComplex0(FancyComplex *ob) {
                return ob->getTitle(); }
        void cpp_FancyComplex_print__12FancyComplex0(FancyComplex *ob) {
                ob->print(); }
}
```

7.3 An example with templates

In this sample, we use a template list written in C++.

When you want to use a C++ code which defines or uses template classes or functions,you have to write some code before launching the generation.

1. For each template parameter you have to write a C++ abstract class which contains any operation made on that type in the template class.
   In the following example it gives the classes Node_Param_T_Cat and MyList_Param_T_Cat.
   Create a C++ class which inherits from these abstract classes. (MyFloat in our example)
2. And of course write your Aldor program using this class as parameter.

If you intend to use template classes with constant parameters (template ¡int n¿), there are explanations in 5.5.2.

7.3.1 C++ header file

```
#define null 0

template <class T>
class Node {

  T *value;
  Node<T> *next;

 public:
  Node<T> (T *val);
  T *getValue();
  Node<T> *getNext();
  void setNext(Node<T> *n);
};


template <class T>
class MyList {

  Node<T> *head;
  Node<T> *current;
  Node<T> *last;

 public:
  MyList();
  void    addAtEnd(T *val);
  void    rewind();
```

```
  Node<T> *getCurrent();
  Node<T> *gotoNext();
  char * toString();
};
/* ************************ */
class Node_Param_T_Cat{
  public:
    virtual char * toString(){}

};

class MyList_Param_T_Cat : public Node_Param_T_Cat{
};
/* ************************ */

class MyCell : public MyList_Param_T_Cat{
  float  f;
  double d;

 public:
  MyCell(float ff, double dd);
  char * toString();
};
```

7.3.2 Aldor test file

```
#include "t1_as.as"

f():() == {
        import from MyList(MyCell),MyCell, DoubleFloat, SingleInteger;
        import from CppSingleFloat,CppDoubleFloat;

        ml : MyList(MyCell):= [];

        addAtEnd(ml,[[-1.5],[3.98567]]);
        print <<" ===> LIST=" << toString(ml) << newline;
        addAtEnd(ml,[[-2.5],[3454.97]]);
        print <<" ===> LIST=" << toString(ml) << newline;
        addAtEnd(ml,[[-3.5],[-7533.96]]);
        print <<" ===> LIST=" << toString(ml) << newline;
        addAtEnd(ml,[[-4.5],[0.0]]);
        print <<" ===> LIST=" << toString(ml) << newline;

        print << newline;

}


f()
```

7.3.3 Generation step by step

Instead of the following lines you can also type make.

```
c++2aldor t1.C t1
axiomxl -Fx -C link=g++ user.as *.o
```

7.3.4 Output

```
 ===> LIST=[ -1.500000 3.985670 ]
 ===> LIST=[ -1.500000 3.985670 ] [ -2.500000 3454.970000 ]
 ===> LIST=[ -1.500000 3.985670 ] [ -2.500000 3454.970000 ]
           [ -3.500000 -7533.960000 ]
 ===> LIST=[ -1.500000 3.985670 ] [ -2.500000 3454.970000 ]
           [ -3.500000 -7533.960000 ] [ -4.500000 0.000000 ]
```

7.3.5 Aldor generated code

```
#include "axllib.as"
import from Machine, SingleInteger;
#library CppTypes "CppTypes__as.ao"
import from CppTypes;
define Node__Cat(T: Node__Param__T__Cat): Category ==  with {
        getValue: (%) -> T;
        getNext: (%) -> Node(T pretend Node__Param__T__Cat);
        setNext: (%,Node(T pretend Node__Param__T__Cat)) -> ();
}
Node(T: Node__Param__T__Cat): Node__Cat(T) with {
        bracket: (T) -> %;
} == add {
        Rep ==> Record (value: T,next: Node(T pretend Node__Param__T__Cat));
        import from Rep;
        bracket (parm0: T) : % == cpp__create______t4Node1ZX01PX010(T,parm0);
        getValue (ob: %) : T == cpp__Node__getValue____t4Node1ZX010(T,ob);
        getNext (ob: %) : Node(T pretend Node__Param__T__Cat)
           == cpp__Node__getNext____t4Node1ZX010(T,ob);
        setNext (ob: %,parm1: Node(T pretend Node__Param__T__Cat)) : ()
           == cpp__Node__setNext____t4Node1ZX01Pt4Node1ZX010(T,ob,parm1);
}
define MyList__Cat(T: MyList__Param__T__Cat): Category ==  with {
        addAtEnd: (%,T) -> ();
        rewind: (%) -> ();
        getCurrent: (%) -> Node(T pretend Node__Param__T__Cat);
        gotoNext: (%) -> Node(T pretend Node__Param__T__Cat);
        toString: (%) -> String;
}
MyList(T: MyList__Param__T__Cat): MyList__Cat(T) with {
        bracket: () -> %;
} == add {
        Rep ==> Record (head: Node(T pretend Node__Param__T__Cat),current: Node(
T pretend Node__Param__T__Cat),last: Node(T pretend Node__Param__T__Cat));
        import from Rep;
        bracket () : % == cpp__create______t6MyList1ZX010(T);
        addAtEnd (ob: %,parm2: T) : ()
           == cpp__MyList__addAtEnd____t6MyList1ZX01PX010(T,ob,parm2);
        rewind (ob: %) : () == cpp__MyList__rewind____t6MyList1ZX010(T,ob);
        getCurrent (ob: %) : Node(T pretend Node__Param__T__Cat)
           == cpp__MyList__getCurrent____t6MyList1ZX010(T,ob);
        gotoNext (ob: %) : Node(T pretend Node__Param__T__Cat)
```

```
            == cpp__MyList__gotoNext____t6MyList1ZX010(T,ob);
        toString (ob: %) : String == cpp__MyList__toString____t6MyList1ZX010(T,ob);
}
define Node__Param__T__Cat: Category ==  with {
        toString: (%) -> String;
}
define MyList__Param__T__Cat: Category == Node__Param__T__Cat with {
        -- Inheritance from Node_Param_T_Cat --
        default {
           toString (ob: %) : String
              == cpp__Node__Param__T__Cat__toString____16Node__Param__T__Cat0(
                 ob pretend Node__Param__T__Cat);
        }
}
define MyCell__Cat: Category == MyList__Param__T__Cat with {
        -- Inheritance from MyList_Param_T_Cat --
        default {
        }
        toString: (%) -> String;
}
MyCell: MyCell__Cat with {
        bracket: (CppSingleFloat,CppDoubleFloat) -> %;
} == add {
        Rep ==> Record (f: CppSingleFloat,d: CppDoubleFloat);
        import from Rep;
        bracket (parm3: CppSingleFloat,parm4: CppDoubleFloat) : %
           == cpp__create______6MyCellfd0(parm3,parm4);
        toString (ob: %) : String == cpp__MyCell__toString____6MyCell0(ob);
}
div__t: Type == add;
ldiv__t: Type == add;
lldiv__t: Type == add;
size__t: Type == add;
uid__t: Type == add;
FILE: Type == add;
__G__dev__t: Type == add;
__G__fpos__t: Type == add;
__G__gid__t: Type == add;
__G__ino__t: Type == add;
__G__mode__t: Type == add;
__G__nlink__t: Type == add;
__G__off__t: Type == add;
__G__pid__t: Type == add;
__G__ptrdiff__t: Type == add;
```

```
__G__sigset__t: Type == add;
__G__size__t: Type == add;
__G__time__t: Type == add;
__G__uid__t: Type == add;
__G__wchar__t: Type == add;
__G__ssize__t: Type == add;
__G__wint__t: Type == add;
__G__va__list: Type == add;
__G__int8__t: Type == add;
__G__uint8__t: Type == add;
__G__int16__t: Type == add;
__G__uint16__t: Type == add;
__G__int32__t: Type == add;
__G__uint32__t: Type == add;
__G__int64__t: Type == add;
__G__uint64__t: Type == add;
__IO__FILE: Type == add;
streampos: Type == add;
streamsize: Type == add;
____fmtflags: Type == add;
____iostate: Type == add;
__ios__fields: Type == add;
Init: Type == add;
ios: Type == add;
__seek__dir: Type == add;
streammarker: Type == add;
streambuf: Type == add;
filebuf: Type == add;
ostream: Type == add;
istream: Type == add;
iostream: Type == add;
__IO__istream__withassign: Type == add;
__IO__ostream__withassign: Type == add;
Iostream__init: Type == add;

import {

        -- methods from class Node
        cpp__create______t4Node1ZX01PX010: (T: Node__Param__T__Cat,T)-> Node(T);
        cpp__Node__getValue____t4Node1ZX010: (T: Node__Param__T__Cat,Node(T))-> T;
        cpp__Node__getNext____t4Node1ZX010: (T: Node__Param__T__Cat,Node(T))
           -> Node(T pretend Node__Param__T__Cat);
        cpp__Node__setNext____t4Node1ZX01Pt4Node1ZX010: (T: Node__Param__T__Cat,
           Node(T),Node(T pretend Node__Param__T__Cat)) -> ();
```

```
        -- methods from class MyList
        cpp__create_____t6MyList1ZX010: (T: MyList__Param__T__Cat)-> MyList(T);
        cpp__MyList__addAtEnd____t6MyList1ZX01PX010: (T: MyList__Param__T__Cat,
          MyList(T),T) -> ();
        cpp__MyList__rewind____t6MyList1ZX010: (T: MyList__Param__T__Cat,MyList(T))
          -> ();
        cpp__MyList__getCurrent____t6MyList1ZX010: (T: MyList__Param__T__Cat,
          MyList(T)) -> Node(T pretend Node__Param__T__Cat);
        cpp__MyList__gotoNext____t6MyList1ZX010: (T: MyList__Param__T__Cat,
          MyList(T)) -> Node(T pretend Node__Param__T__Cat);
        cpp__MyList__toString____t6MyList1ZX010: (T: MyList__Param__T__Cat,
          MyList(T)) -> String;

        -- methods from class Node_Param_T_Cat
        cpp__Node__Param__T__Cat__toString____16Node__Param__T__Cat0:
           (Node__Param__T__Cat) -> String;
        cpp__create__Node__Param__T__Cat0: ()-> Node__Param__T__Cat;

        -- methods from class MyList_Param_T_Cat
        cpp__create__MyList__Param__T__Cat0: ()-> MyList__Param__T__Cat;

        -- methods from class MyCell
        cpp__create_____6MyCellfd0: (CppSingleFloat,CppDoubleFloat)-> MyCell;
        cpp__MyCell__toString____6MyCell0: (MyCell) -> String;
} from Foreign C;
```

.....

7.3.6 C++ generated code

.....

```
#include "t1.C"
#include "CppTypes.H"
extern "C" {

/* methods from class Node */
        Node<Node_Param_T_Cat>* cpp_create___t4Node1ZX01PX010(void *parm5,
          Node_Param_T_Cat* parm0) {
                return new Node<Node_Param_T_Cat>(parm0); }
        Node_Param_T_Cat* cpp_Node_getValue__t4Node1ZX010(void *parm6,
          Node<Node_Param_T_Cat> *ob) {
                return ob->getValue(); }
        Node<Node_Param_T_Cat>* cpp_Node_getNext__t4Node1ZX010(void *parm7,
```

```
          Node<Node_Param_T_Cat> *ob) {
                return ob->getNext(); }
        void cpp_Node_setNext__t4Node1ZX01Pt4Node1ZX010(void *parm8,
          Node<Node_Param_T_Cat> *ob,Node<Node_Param_T_Cat>* parm1) {
                ob->setNext(parm1); }

/* methods from class MyList */
        MyList<MyList_Param_T_Cat>* cpp_create___t6MyList1ZX010(void *parm9) {
                return new MyList<MyList_Param_T_Cat>(); }
        void cpp_MyList_addAtEnd__t6MyList1ZX01PX010(void *parm10,
           MyList<MyList_Param_T_Cat> *ob,MyList_Param_T_Cat* parm2) {
                ob->addAtEnd(parm2); }
        void cpp_MyList_rewind__t6MyList1ZX010(void *parm11,
           MyList<MyList_Param_T_Cat> *ob) {
                ob->rewind(); }
        Node<MyList_Param_T_Cat>* cpp_MyList_getCurrent__t6MyList1ZX010(
          void *parm12,MyList<MyList_Param_T_Cat> *ob) {
                return ob->getCurrent(); }
        Node<MyList_Param_T_Cat>* cpp_MyList_gotoNext__t6MyList1ZX010(void *parm13,
          MyList<MyList_Param_T_Cat> *ob) {
                return ob->gotoNext(); }
        char* cpp_MyList_toString__t6MyList1ZX010(void *parm14,
          MyList<MyList_Param_T_Cat> *ob) {
                return ob->toString(); }

/* methods from class Node_Param_T_Cat */
        char* cpp_Node_Param_T_Cat_toString__16Node_Param_T_Cat0(
           Node_Param_T_Cat *ob) {
                return ob->toString(); }
        Node_Param_T_Cat* cpp_create_Node_Param_T_Cat0() {
                return new Node_Param_T_Cat(); }

/* methods from class MyList_Param_T_Cat */
        MyList_Param_T_Cat* cpp_create_MyList_Param_T_Cat0() {
                return new MyList_Param_T_Cat(); }

/* methods from class MyCell */
        MyCell* cpp_create___6MyCellfd0(CppSingleFloat * parm3,
           CppDoubleFloat *parm4) {
                return new MyCell((float) *parm3,(double) *parm4); }
        char* cpp_MyCell_toString__6MyCell0(MyCell *ob) {
                return ob->toString(); }
}
```

7.4 An example from the PoSSo Library

We took the C++ test file named IterTest2 in the directory lib/Algebra/Poly and converted it in Aldor.

7.4.1 C++ modified file: IterTest2.C

The modifications are the removing of the main function and the addition of Set_Param_T_Cat and CmmObject_Param_T_Cat (in ForTemplates.C). The file, so modified has been named IterTest2b.C

```
#include "ForTemplates.C"

//  $Id: IterTest2.C,v 1.5 1997/10/24 08:19:54 zanoni Exp $

#ifndef lint
static char vcid[] = "$Id: IterTest2.C,v 1.5 1997/10/24 08:19:54 zanoni Exp $";
#endif /* lint */
// File:     IterTest2.C - Polynomial iterators test program -*- C++ -*-
// Package:  PoSSo Polynomial Library.
// Created:  1995 May 12
// Author:   Giovanni A. Cignoni
// E-Mail:   cygnus@di.unipi.it
// Address:  Dipartimento di Informatica
//           Universita' di Pisa
//           Corso Italia 40
//           56125 Pisa
//           ITALY
//
// Copyright (C) 1995 Dipartimento di Matematica, POSSO ESPRIT BRP 6846.
//
// This file is part of the POSSO LIBRARY.
//
// This code is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
//
//



#include "GCoeff.H"

// Use GCoeff instead of BCoeff - Giovanni.
```

```
#define PL_BCoeffRing           PL_GCoeffRing
#define createBCoeffRing        createGCoeffRing


#include "ZpCoeff.H"
#include "SCoeff.H"


#include "DensePP.H"
#include "TDensePP.H"


#include "Polynomial.H"



void outData( PL_PolyIter& i ) {

  PL_Monomial& m = createMonomial();
  PL_Poly& p     = createPoly();

  m = i.current();
  p = i.poly();

  cout << "  polynomial  = ";
  cout << p;
  cout << ";" << endl;

  cout << "  iterator at = ";
  // cout << m;
  cout << i.current();
  cout << "." << endl;
}

#ifdef PL_CMM_TEMPHEAP
  int UseTempHeap = true;
  int HeapSize = 100;
#endif


#ifdef MAIN_IS_REMOVED_FOR_CPP2ALDOR
int main() {

  int nv = 0;
```

```
int nd = 0;

int buff[1024];


// Input PPMonoid specifications.

cout << endl;
cout << "Input PPMonoid specifications ..." << endl;

if( readdirs(cin, nv, nd, buff) == -1 ) {

  cout << "PPMonoid error." << endl;
  exit(1);
}


// Create a CoeffRing and a PPMonoid.

PL_CoeffRing& cr = createBCoeffRing();
PL_PPMonoid& ppm = createTDensePPMonoid(nv, nd, buff);

cout << "Using PPMonoid:" << endl << ppm;


// Create the PolyRing.

PL_PolyRing& pr = createPolyRing(cr, ppm);

setPolyRing(pr);

PL_Monomial& m = createMonomial();

PL_Poly& p1 = createPoly();
PL_Poly& p2 = createPoly();


// Input test data.

cout << endl;
cout << "Input a monomial (using BCoeff and TDensePP in xyzt) ..." << endl;
cin >> m;
cout << "Read m = " << m << "." << endl;
```

```
cout << endl;
cout << "Input a polynomial (using BCoeff and TDensePP in xyzt) ..." << endl;
cin >> p1;
cout << "Read p1 = " << p1 << "." << endl;

cout << endl;
cout << "Input an other polynomial ..." << endl;
cin >> p2;
cout << "Read p2 = " << p2 << "." << endl;
cout << endl;


PL_Poly& ptmp = createPoly(p1);
PL_PolyIter& itmp = createPolyIter(ptmp);


// Go with tests ...

cout << endl;
cout << "Reset the iterator:" << endl;
ptmp = p1;
itmp.reset(ptmp);
outData(itmp);

cout << endl;
cout << "Insert monomial m = " << m << ":" << endl;
itmp += m;
outData(itmp);

ptmp = p1;
itmp.reset(ptmp);
cout << "Append monomial m = " << m << ":" << endl;
itmp.append(m);
outData(itmp);

ptmp = p1;
itmp.reset(ptmp);
cout << "Insert polynomial p2 = " << p2 << ":" << endl;
itmp += p2;
outData(itmp);

ptmp = p1;
itmp.reset(ptmp);
cout << "Append polynomial p2 = " << p2 << ":" << endl;
```

```
itmp.append(p2);
outData(itmp);
cout << endl;


if( !p1.isZero() ) {

  cout << endl;
  cout << "Reset the iterator and advance it:" << endl;
  ptmp = p1;
  itmp.reset(ptmp);
  ++itmp;
  outData(itmp);

  cout << endl;
  cout << "Insert monomial m = " << m << ":" << endl;
  itmp += m;
  outData(itmp);

  ptmp = p1;
  itmp.reset(ptmp);
  ++itmp;
  cout << "Append monomial m = " << m << ":" << endl;
  itmp.append(m);
  outData(itmp);

  ptmp = p1;
  itmp.reset(ptmp);
  ++itmp;
  cout << "Insert polynomial p2 = " << p2 << ":" << endl;
  itmp += p2;
  outData(itmp);

  ptmp = p1;
  itmp.reset(ptmp);
  ++itmp;
  cout << "Append polynomial p2 = " << p2 << ":" << endl;
  itmp.append(p2);
  outData(itmp);
  cout << endl;


  cout << endl;
  cout << "Reset the iterator and go to last:" << endl;
```

```
    ptmp = p1;
    itmp.reset(ptmp);
    itmp.goToLast();
    outData(itmp);

    cout << endl;
    cout << "Insert monomial m = " << m << ":" << endl;
    itmp += m;
    outData(itmp);

    ptmp = p1;
    itmp.reset(ptmp);
    itmp.goToLast();
    cout << "Append monomial m = " << m << ":" << endl;
    itmp.append(m);
    outData(itmp);

    ptmp = p1;
    itmp.reset(ptmp);
    itmp.goToLast();
    cout << "Insert polynomial p2 = " << p2 << ":" << endl;
    itmp += p2;
    outData(itmp);

    ptmp = p1;
    itmp.reset(ptmp);
    itmp.goToLast();
    cout << "Append polynomial p2 = " << p2 << ":" << endl;
    itmp.append(p2);
    outData(itmp);
    cout << endl;
  }

}
#endif
```

7.4.2 Aldor test file

```
#include "axllib.as"

#include "IterTest2_as.as";

SI ==> SingleInteger;

import {
        cpp__apply__cin: () -> istream;
        cpp__apply__cout: () -> ostream;
} from Foreign C;

apply(g: 'Global',c: 'cin'): istream == cpp__apply__cin();
apply(g: 'Global',c: 'cout'): ostream == cpp__apply__cout();


itertest(): () == {
  import from SI, 'Global', 'cin', 'cout';

  local nv: PrimitiveArray(SI) := new (1,0);
  local nd: PrimitiveArray(SI) := new (1,0);

  local buff: PrimitiveArray(SI) := new(1024,0);

  -- Input PPMonoid specifications.

  print << newline;
  print << "Input PPMonoid specifications ..." << newline;

  if (readdirs(Global.cin, nv, nd, buff) = -1) then {

    print << "PPMonoid error." << newline;
    return;
  }
  -- Create a CoeffRing and a PPMonoid.

  cr: PL__CoeffRing := createGCoeffRing();
  ppm: PL__PPMonoid  := createTDensePPMonoid(nv.1, nd.1, buff);

  print << "Using PPMonoid:" << newline;
  Global.cout << ppm;

  -- Create the PolyRing.
```

```
pr: PL__PolyRing := createPolyRing(cr, ppm);

setPolyRing(pr);

m: PL__Monomial := createMonomial();

p1: PL__Poly := createPoly();
p2: PL__Poly := createPoly();


++ Input test data.

print << newline;
print << "Input a monomial (using BCoeff and TDensePP in xyzt) ..." << newline;
Global.cin >> m;
print << "Read m = ";
Global.cout << m;
print << "." << newline;

print << newline;

print << "Input a polynomial (using BCoeff and TDensePP in xyzt) ..." << newline;
Global.cin >> p1;
print << "Read p1 = ";
Global.cout << p1;
print << "." << newline;

print << newline;
print << "Input an other polynomial ..." << newline;
Global.cin >> p2;
print << "Read p2 = ";
Global.cout << p2;
print << "." << newline;
print << newline;


ptmp: PL__Poly := createPoly(p1);
itmp: PL__PolyIter := createPolyIter(ptmp);

-- Go with tests ...

print << newline;
print << "Reset the iterator:" << newline;
```

```
    print << newline;
    print << "Insert monomial m = ";
    Global.cout <<  m;
    print << ":" << newline;
    C_+_=(itmp,m);
    outData(itmp);

    C_=(ptmp,p1);
    reset(itmp,ptmp);
    goToLast(itmp);
    print << "Append monomial m = ";
    Global.cout << m;
    print << ":" << newline;
    append(itmp,m);
    outData(itmp);

    C_=(ptmp,p1);
    reset(itmp,ptmp);
    goToLast(itmp);
    print << "Insert polynomial p2 = ";
    Global.cout << p2;
    print << ":" << newline;


    C_+_=(itmp,p2);
    outData(itmp);

    C_=(ptmp,p1);
    reset(itmp,ptmp);
    goToLast(itmp);
    print << "Append polynomial p2 = ";
    Global.cout << p2;
    print << ":" << newline;
    append(itmp,p2);
    outData(itmp);
    print << newline;

  }

}

itertest()
```

7.4.3 Generation step by step

In this paragraph, we explain command by command the steps to follow in order to make it work. The file HOWTO also describes the following steps.

In standard mode:

1. Edit the excludes.lst to have the standard header files from gcc and libg++ generated shortly (note: if you don't do that, you might encounter some errors). To do so, put the name of the root directories of these header files.

2. Edit Makefile and set :
   - POSSO_ROOT to the location of the PoSSoLib-2.1.9
   - GXX_INCLUDES to the location of you c++ standard library

3. make no_split

In split mode:

1. Edit the excludes.lst to have the standard header files from gcc and libg++ generated shortly (note: if you don't do that, you might encounter some errors). To do so, put the name of the root directories of these header files.

2. Edit Makefile and set :
   - POSSO_ROOT to the location of the PoSSoLib-2.1.9
   - GXX_INCLUDES to the location of you c++ standard library

3. Add the following lines in the library header files:
   $(POSSO_ROOT)/installed/include
   This is to avoid having to modify the generated code to make it compile.

   ```
   in Monomial.H  (just after #define PL_Monomial_H)
        #include <cmm.h>
        #include <Coeff.H>
        #include <PP.H>

   in gmp.h  (at the beginning)
        #include <libio.h>
   ```

4. Edit xml2as.cfg and change the directory for the PoSSo library.
   example:

   ```
   @blocks
   /home/fdefaix/frisco/PoSSoLib-2.1.9/installed/include/PolyIter.H
   /home/fdefaix/frisco/PoSSoLib-2.1.9/installed/include/PolyRing.H
   ```

```
/home/fdefaix/frisco/PoSSoLib-2.1.9/installed/include/Poly.H

@extblocks
.inl.H
.inl
```

5. make split

7.4.4 Output

When running the program 'user < testI2-2.in' ('possoUser < testI2-2.in' in split mode) :

```
Input PPMonoid specifications ...
Using PPMonoid:
{ 8,
  { N, { a, b, c, d, e, f, g, h } },
  { M, { 0, 0, 0, 0, 0, 0, 0, 0 } },
  { W, { 1, 1, 1, 1, 1, 1, 1, 1 } },
  { S, { 1, 1, 1, 1, 1, 1, 1, 1 } },
  { E, { 1, 1, 1, 1, 1, 1, 1, 1 } },
  { U, { 1, 1, 1, 1, 1, 1, 1, 1 } },
  { L, { 1, 1, 1, 1, 1, 1, 1, 1 } }
}

Input a monomial (using BCoeff and TDensePP in xyzt) ...
Read m = 1e.

Input a polynomial (using BCoeff and TDensePP in xyzt) ...
Read p1 =  a + b + c + d.

Input an other polynomial ...
Read p2 =  f + g + h.


Reset the iterator:
  polynomial  =  a + b + c + d;
  iterator at = 1a.

Insert monomial m = 1e:
  polynomial  =  e + a + b + c + d;
  iterator at = 1a.
Append monomial m = 1e:
  polynomial  =  a + e + b + c + d;
  iterator at = 1a.
Insert polynomial p2 =  f + g + h:
  polynomial  =  f + g + h;
  iterator at = 1f.
Append polynomial p2 =  f + g + h:
  polynomial  =  a + f + g + h;
  iterator at = 1a.
```

```
Reset the iterator and advance it:
  polynomial  =  a + b + c + d;
  iterator at = 1b.

Insert monomial m = 1e:
  polynomial  =  a + e + b + c + d;
  iterator at = 1b.
Append monomial m = 1e:
  polynomial  =  a + b + e + c + d;
  iterator at = 1b.
Insert polynomial p2 =  f + g + h:
  polynomial  =  a + f + g + h;
  iterator at = 1f.
Append polynomial p2 =  f + g + h:
  polynomial  =  a + b + f + g + h;
  iterator at = 1b.


Reset the iterator and go to last:
  polynomial  =  a + b + c + d;
  iterator at = 1d.

Insert monomial m = 1e:
  polynomial  =  a + b + c + e + d;
  iterator at = 1d.
Append monomial m = 1e:
  polynomial  =  a + b + c + d + e;
  iterator at = 1d.
Insert polynomial p2 =  f + g + h:
  polynomial  =  a + b + c + f + g + h;
  iterator at = 1f.
Append polynomial p2 =  f + g + h:
  polynomial  =  a + b + c + d + f + g + h;
  iterator at = 1d.
```

7.4.5 Aldor generated code

```
.....

This file is to big to be included here.
Please refer to the file 'iter_as.as'.

.....
```

7.4.6 C++ generated code

```
.....

This file is to big to be included here.
Please refer to the file 'iter_cc.C'.

.....
```

Chapter 8

Revision History

Here is an history of the "C++ to *Aldor*" application since the release 1.0

8.1 V1.3c to V1.3d

- Updated PoSSo example for PoSSo-2.1.10
- Update for AIX
- Known Bugs fixed

8.2 V1.3b to V1.3c

- Known Bugs fixed
- Updated PoSSo example
- Optimization of compilation in split mode

8.3 V1.3 to V1.3b

- Known Bugs fixed
- Better array handling
- New flexible options for `cpp2aldor`

8.4 V1.2 to V1.3

- Most basic types are handled (but long double)
- Generation of Aldor can be split (use of libraries)
- Installation and use with a configuration file

8.5 V1.1c to V1.2

- Accessors to static variables
- Better template class handling
- Template function handling
- Prettier name for operators as global functions
- Nested classes

8.6 V1.1 to V1.1c

- Sources reorganisation
- Addition of comments

8.7 V1.0 to V1.1

- Integration of cpp2xml in gcc as a XML option
- Stubs for global variables and functions
- Prettier names for operators
- A new example from the Posso-lib

Part II

Using *Aldor* from C++

It may be very useful to be able to use *Aldor* objects in C++. For example as *Aldor* develops a great interaction with the computer-algebra libraries from AXIOM, *Aldor* can be used as an intermediate language between C++ and AXIOM. Furthermore the language *Aldor* has been based on strong functional idioms (e.g dependent types, high-level functions, functions as first values, ...), programming using two different languages can be intricate but also rewarding.

On one hand, C++ provides a very good interface with C and is widely used (thus enabling people to feel comfortable with a "known" language). On the other hand, *Aldor* provides powerful concepts inherited from functional and higher order programming and thus enables the programmer to create advanced libraries.

This part describes the new feature of the *Aldor* compiler which allows for code generation in order to use *Aldor* types in C++.

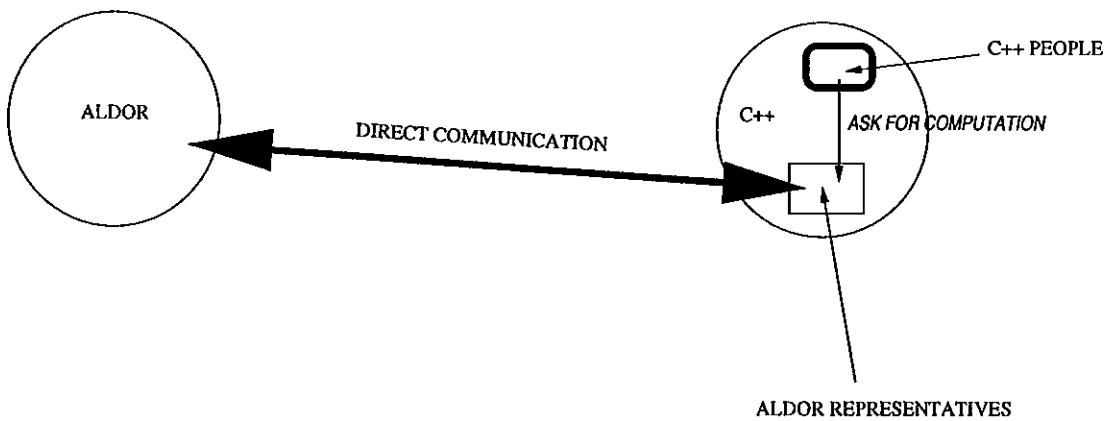
Chapter 9

Using the Tool

9.1 General presentation

To call *Aldor* pieces of code from a C++ program, we first need to understand the general scheme. We have two worlds: *Aldor* and C++. In our case the representatives of the world called *Aldor* (i.e categories and domains) are sent to the C++ world as classes (let's say they are transformed during the trip).

When people from the world C++ want some work to be done by people from the world *Aldor*, they just "order" the work to the representatives. The order is transmitted to people from the world *Aldor* who take care of performing the needed work (generally computations). They send back the results to the representatives who transmit them to C++ people. Schematically it gives:



9.2 Example

This section looks at a simple example in order to show how the whole process of the translation is working. First examine the following *Aldor* types that we want to interface:

```
SI ==> SingleInteger
```

```
define EncapsSICat: Category == with {
        bracket: SI -> %;
        plus:    (%,%) -> %;
        minus:   (%,%) -> %;
        getVal:  % -> SI;
}

EncapsSI: EncapsSICat == add {
        Rep ==> Record(i: SI);
        import from Rep;

        bracket(val: SI):     % == per [val];
        plus(ob1: %,ob2: %):  % == per [rep(ob1).i + rep(ob2).i];
        minus(ob1: %,ob2: %): % == per [rep(ob1).i - rep(ob2).i];
        getVal(ob: %):       SI == rep(ob).i;
}
```

Let's assume we have the following files:

- `Encaps.as` which contains the definition of the above types
- `AnotherCode.as` which is a complementary file which may be used for the implementation of `Encaps.as`
- `MyCodeUsingAldor.C` which will make use of the types and which contains the 'main' function
- `MiscStuff.C` more C++ code ...

The main goal is: we need the C++ version of the *Aldor* types in order to use them in a C++ application. The basic scheme is thus the following: translate the *Aldor* types into C++ classes, compile the result and link with the C++ application using these types.
Now, here is what you have to do:
First, let's compile the miscellaneous code:

```
$ g++ -c MiscStuff.C
$ axiomxl -Fo AnotherCode.as
```

Now we need to generate the C++ code corresponding to `Encaps.as`. We will use the modified version of the *Aldor* compiler:

```
axiomxl -Fc++ Encaps.as
```

This will generate two files: `Encaps_as.as` and `Encaps_cc.h`. The file `Encaps_as.as` (that we have to compile to an object file: `Encaps_as.o`) contains the stubs for the *Aldor* functions and the
`export ... to Foreign C;` statement. The file `Encasp_cc.h` contains the interface to the *Aldor* types as C++ classes:

.....

```
template <class PercentType>
class EncapsSICat {
public:
   virtual ~EncapsSICat() {}
   virtual PercentType * plus(PercentType *) = 0;
   virtual PercentType * minus(PercentType *) = 0;
   virtual int  getVal() = 0;
};
class EncapsSI:  virtual public EncapsSICat<EncapsSI> {
   EncapsSI *ptr;
public:
   EncapsSI() { ptr = 0; }
   EncapsSI(EncapsSI *orig) { ptr = orig; }
   EncapsSI(void *orig) {
       ptr = reinterpret_cast<EncapsSI *>(orig);
   }
   virtual ~EncapsSI() {}
   friend EncapsSI *realObject(EncapsSI *ob) {
       return ob->ptr;
   }
   static void *givetype() {
       return ALDOR_givetype_EncapsSI();
   }

   static EncapsSI * bracket(int parm0) {
       return new EncapsSI(ALDOR_bracket_EncapsSI0(parm0));
   }
   virtual EncapsSI * plus(EncapsSI * parm0) {
       return new EncapsSI(
         ALDOR_plus_EncapsSI1(ptr,realObject(parm0)));
   }
   virtual EncapsSI * minus(EncapsSI * parm0) {
       return new EncapsSI(
         ALDOR_minus_EncapsSI2(ptr,realObject(parm0)));
   }
   virtual int  getVal() {
       return ALDOR_getVal_EncapsSI3(ptr);
   }
};
```

We will come back to the meaning of all of this in the next chapter. Now to get an executable file, we have to link together everything. Note that the main file `MyCodeUsingAldor.C` should contain a `#include "Encaps_cc.h"` instruction to be allowed to use the types. The command to compile everything is:

```
g++ -o application MyCodeUsingAldor.C Encaps_as.o MiscStuff.o \
    -I$AXIOMXLROOT/include -L$AXIOMXLROOT/lib -laxllib -lfoam -lm
```

Let's summarize all the actions that have been performed here:

1. `g++ -c MiscStuff.C` and `axiomxl -Fo AnotherCode.as` to get the object files of the miscellaneous code.
2. `axiomxl -Fc++ Encaps.as` to translate the code we want to interface.
3. `axiomxl -Fo Encaps_as.as` to get the object file from the *Aldor* stubs
4. `g++ -o application MyCodeUsingAldor.C Encaps_as.o MiscStuff.o \`
   `-I$AXIOMXLROOT/include -L$AXIOMXLROOT/lib`
   `-laxllib -lfoam -lm`
   to compile the whole application

9.3 Options

An option '-P' has been added to the *Aldor* compiler to control the C++ generation. Using '-P', the user is allowed to ask for the following features:

- **basicfile=¡bf¿** this specifies the name of a file containing the basic type correspondence. If nothing is specified, the file `basic.typ` is searched in the current directory and then in `$AXIOMXLROOT/include`. If no file can be found a hard-coded correspondence is used.
- **discrim-return** is used to handle overloading on the return type only, which is not available in C++. To manage this concept, we add the name of the return type to the

function identifier. As the result of the generation may be difficult to use, the user has to specify this option. The corresponding option **no-discrim-return** is the default. If the source file contains overloaded functions on the return type only, the result of the generation won't compile.

Chapter 10

Our Model

This chapter explains the basic rules used for this tool. To get an idea of the object model and the type correspondence please refer to (4).

10.1 Basic types

To map *Aldor* basic types to C++ basic types, we are going to use what has been defined in 4. The mapping will thus be the following:

Aldor type	C++ type
`SingleInteger`	`int`
`Character`	`char`
`String`	`char*`
`HalfInteger`	`short`
`SingleFloat`	`float`
`DoubleFloat`	`double*`
`Boolean`	`bool`
`Pointer`	`void*`
`A`	`A*`

Special types like `List`, `Record`, `Array` and so on are currently translated by `void *`.

This mapping is not hard coded. It is possible to provide a file called `basic.typ` which will contain the mapping of basic types. The format of the file is very simple, each line describe one correspondence: `C++_type Aldor_type`. For example, the file for the correspondence above is:

```
int SingleInteger
char Character
char* String
short HalfInteger
float SingleFloat
double* DoubleFloat
bool Boolean
```

`void* Pointer`

Notes:

- we didn't include `A  A*`, because A here is not a basic type but it is a general rule of translation: an *Aldor*type corresponds to a pointer on a C++ object.
- we use `void*` and not `void *`, the parser of this file is sensitive to this kind of blanks.

10.2 Naming of functions and operators

When it is possible to get a direct mapping from an *Aldor* operator to a C++ operator (for example it is possible with +) we translate `+: (%,%)->%;` to `PercentType *operator+(PercentType *)`. When the operators are not common to the languages, the translation uses the naming used by the *Aldor* compiler to translate *special* characters in C. For example, `set!` is translated to `set__BANG__`. Any function containing a special symbol or any *Aldor* operator which doesn't translate to a C++ operator is generated using this convention.

Here is the table of correspondence:

Symbol	Translation	Symbol	Translation
!	_BANG_	;	_SEMI_
"	_QUOTE_	<	_LT_
#	_SHARP_	=	_EQ_
$	_DOLLR_	>	_GT_
%	_PCENT_	?	_QMARK_
&	_AMPER_	@	_AT_
'	_APOS_	[	_OBRACK_
(	_OPAREN_	\	_BSLSH_
)	_CPAREN_	]	_CBRACK_
*	_STAR_	^	_HAT_
+	_PLUS_	_	__
,	_COMMA_	`	_GRAVE_
-	_MINUS_	{	_OBRACE_
.	_DOT_	|	_BAR_
/	_SLASH_	}	_CBRACE_
:	_COLON_	~	_TILDE_

10.3 Category/Abstract class Correspondence

An *Aldor* category (parameterized or not) is translated into a template abstract class. The first template parameter (and maybe the only one if the category was not parameterized) is `class PercentType`. It is a way to translate the *Aldor* % type.

The non-member functions (i.e those without a '%' as first parameter) are not included in the definition of the class. All the methods are pure virtual methods.

The inheritance of categories corresponds to a virtual public inheritance between abstract classes in C++.

10.4 Domain/Class Correspondence

The translation of a domain has two parts: an abstract class and a class. The abstract class corresponds to the 'type' of the domain (i.e the category) and the class corresponds to the body of the add statement without the data representation and the private members which are useless for interoperability purposes.

An extra abstract class is created when the 'type' part of the domain definition contains a `with` statement.
For example,

```
A: with {
    foo: %->%;
} == add {...}
```

gets translated to

```
template <class PercentType>
class ACategoryExtra {
public:
    virtual PercentType *foo() = 0;
};
```

and

```
class A: virtual public ACategoryExtra<A> {
     A* ptr; /* pointer to the real Aldor object */
public:
    virtual A *foo() { return ALDORfoo_A(ptr); }
};
```

It is interesting to note that the parameter given to the abstract class is the class A itself.

10.5 Multiple return functions

Aldor functions may return zero, one or several values of different types. Up to now we only showed examples using functions which were returning zero or one value. For instance,

```
f: SI -> SI;
```

gets translated to

```
int f(int);
```

Dealing with multiple return functions with a language that doesn't allow this feature is not so easy. In [W+94] there is an example of the translation of an *Aldor* multiple return function in C. Instead of returning several values, the C function will return none (`void`) and will take extra parameters which correspond to the return values. The difference with regular parameter is that they use an extra indirection to be able to modify the contents of the object. This is a regular C way to return several values. For example in C, a function `f` taking two integers may return three integers:

```
void f(int arg1, int arg2, int *ret1, int *ret2, int *ret3);
```

Aldor handles multiple return functions in the same way. In our translation we need to be aware of that and to create extra code. Indeed if no extra code is generated, the returned

values will be pointers on real *Aldor* objects and not the C++ counterpart needed to run the C++ program. That's why we first need to declare some local variables in the stubs which will get the pointers to the real *Aldor* objects. After the function call, we create the new objects from the values created by the function.
In Aldor: `foo: (DomA, DomB) -> (DomC, DomD);`
In C++, the stub will look like:

```
void foo(DomA *parm0, DomB *parm1, DomC **ret0, DomD **ret1) {
    DomC *local0; DomD *local1;
    ALDOR_foo(parm0, parm1, &local0, &local1);
    *ret0 = new DomC(local0); *ret1 = new DomD(local1);
}
```

10.6 Some explanations about the example

Let's take a look at the example of the previous chapter. In this section, we will see each interesting or relevant point in the translation of the *Aldor* types.

10.6.1 EncapsSICat

The category `EncapsSICat` is translated as an abstract class, with a special template argument. As explained above this parameter will replace the *Aldor* % which corresponds to the current type. Each class which will derive from this abstract class will give *itself* as template argument.

10.6.2 Header of the class EncapsSI

Now if we look at the translation of the domain, a few things have to be explained. First the header `EncapsSI: EncapsSICat == add ...` corresponds to a **virtual public** inheritance from the abstract class `class EncapsSI: virtual public EncapsSICat<EncapsSI> ...`. Furthermore some members have been added to the class:

- `EncapsSI *ptr;` is a pointer to the *real Aldor* object. It is given as parameter to the stubs.
- `EncapsSI(void *orig);` is used for the same purpose but has to deal with "void *" arguments because the stubs use 'void *' as type.
- `virtual ~EncapsSI() {}` is used to ensure that a virtual function will exist in the class (to allow dynamic casting).
- `friend EncapsSI *realObject(EncapsSI *ob)` enables to access the real *Aldor* object given as parameter to the interfaced functions.
- `static void *givetype();` is used to pass the domain type of an object in case of template handling. For example, the *Aldor* function `f: (T: TCat, T)->();` needs a type parameter (its first argument), the way we chose to do that is to create an *Aldor* function for each domain which returns the domain itself. This function will be called from C++ and the result will be directly given the *Aldor* stub.

10.6.3 Stub functions

The interfaced methods (those which are exported to C and imported from C via the `extern "C"` directive) use a naming of the form: `ALDOR[method name]_[class name]`. The interfaced global functions use a name of the form: `ALDOR[function name]_Global`.

Chapter 11

Limitations

11.1 Limitations

Here is a list of the limitations, features and known bugs:

These items have not been treated and the result of the translation will probably not compile because it is hard to detect and to skip:

- Inner Domains

The following items have not been treated, entities using them are skipped (i.e not interfaced):

- Global variables
- Default parameters
- Macros
- Dependent types
- Types using entities other than domains as their parameter (in particular values are not handled)
- Types using domains defined by inline categories
- Functions using Exit or Generator
- Functions using Literal
- Function pointers in multiple return (e.g `f: (SI,SI) -> (SI,SI->SI)`)
- Function pointers returning multiple values

The limitations are:

- No translation of the standard *Aldor* libraries (axllib and basicmath).
- `void *` is used for the translation of complicated types like 'List X', 'Array X', ... This means that a list may be created by an *Aldor* function and given as parameter to another *Aldor* function but it can't be used in C++.

- We can not call, in C++, functions returned by Aldor functions.

11.2 Features

Let's see now what is done:

- Categories (parameterized or not) are translated as template abstract classes.
- Domains and parameterized domains are now translated to classes and template classes.
- Operators are handled.
- Function pointers are translated in most of the cases.
- Type determination works better (`Domain: Category == AnotherDomain` is now handled).
- Multiple returns are handled (except for values of type function pointers).
- Global functions (i.e not belonging to a type) are translated.

Chapter 12

Examples

12.1 DataBase of Complex values

This first example will give you an idea of how to proceed to do the translation. The *Aldor* source defines two types ItemComplex and ItemPrettyComplex. It also defines a DBComplex type to manipulate a list of ItemComplex. The function f is a test of these types.

Here is the *Aldor* code:

```
#include "axllib.as"

SI ==> SingleInteger;

define BasicItem: Category == with { display: % -> (); }

ItemComplex: BasicItem with {
     bracket: (SI,SI) -> %;
     plus   : (%,%) -> %;
} == add {
     Rep ==> Record(re: SI, im: SI);
     import from Rep;

     bracket(i: SI, j: SI): % == per [i,j];
     display(o: %): () == {
            print << "Value of the complex: (" << rep(o).re;
            print << "," << rep(o).im << ")" << newline;
     }
     plus(o1: %, o2: %): % ==
            per [rep(o1).im + rep(o2).im,
                 rep(o1).re + rep(o2).re];
}

ItemPrettyComplex: BasicItem with {
```

```
        bracket: (SI,SI,String) -> %;
        plus   : (%,%) -> %;
        getName: % -> String;
    } ==  add {
        Rep ==> Record(base: ItemComplex, name: String);
        import from Rep;

        bracket(i: SI, j: SI, n: String): % == per [[i,j],n];
        plus(o1: %, o2: %): % ==
               per [plus(rep(o1).base,rep(o2).base),
                    concat(rep(o1).name," ",rep(o2).name)];
        getName(o: %): String == rep(o).name;
        display(o: %): () == {
            print << "Complex name: " << rep(o).name << newline;
            display(rep(o).base);
        }
    }

    DBComplexCat: Category == with {
        bracket: () -> %;
        insert:  (%,ItemComplex) -> ();
        display: % -> ();
    }

    DBComplex: DBComplexCat == add {
        Rep ==> Record(set: List ItemComplex);
        import from Rep;

        bracket(): % == per [nil];
        insert(o: %, i: ItemComplex): () ==
               rep(o).set := cons(i,rep(o).set);
        display(o: %): () == {
             for i in tails rep(o).set repeat
                    display(i.first);
        }
    }


    f(): () == {
        import from DBComplex, ItemComplex;
        import from SI;

        db: DBComplex := [];
        insert(db,[3,4]);
```

```
    insert(db,[12,4]);
    insert(db,[3,9]);
    insert(db,[0,1]);
    insert(db,[1,32]);
    insert(db,[3,45]);
    insert(db,[63,4]);
    display(db);
}
```

The application written in C++ will first call the function `f`. Then it will create some objects to test the types on the C++ side. At last a class DBPrettyComplex is created in C++ and uses, as a component, the type PrettyComplex defined above in *Aldor*.

Here is the C++ source code:

```
#include <stdio.h>
#include "db_cc.h"

/* ... here definition of a 'List' class ... */

class DBPrettyComplex {
    List<ItemPrettyComplex> *l;
public:
    DBPrettyComplex() { l = new List<ItemPrettyComplex>(); }
    void insert(ItemPrettyComplex *ipc) {
        l->insert(ipc);
    }
    void display() {
        while (!l->Empty()) {
            l->first()->display();
            l->gotoNext();
        }
    }
};


main() {
    DBComplex *db;
    DBPrettyComplex *dbp;

    printf("C++ side calling Aldor function f\n");
    f();
    printf("End of Aldor function f\n\n");

    printf("Now let's use the classes ...\n");
    db = DBComplex::bracket();
    db->insert(ItemComplex::bracket(25,50));
    db->insert(ItemComplex::bracket(30,60));
    db->insert(ItemComplex::bracket(40,80));
    db->insert(ItemComplex::bracket(12,24));
    db->insert(ItemComplex::bracket(8,16));
    db->display();
    printf("End of use of classes\n\n");

    printf("DB pretty complex\n");
    dbp = new DBPrettyComplex();
    dbp->insert(ItemPrettyComplex::bracket(1,2,"A"));
```

```
    dbp->insert(ItemPrettyComplex::bracket(6,3,"B"));
    dbp->insert(ItemPrettyComplex::bracket(12,8,"C"));
    dbp->insert(ItemPrettyComplex::bracket(51,62,"D"));
    dbp->insert(ItemPrettyComplex::bracket(34,87,"E"));
    dbp->insert(ItemPrettyComplex::bracket(231,322,"F"));
    dbp->display();
    printf("End DB pretty complex\n");
}
```

After compilation and linkage we obtain an executable which produces this:

```
C++ side calling Aldor function f
Value of the complex: (63,4)
Value of the complex: (3,45)
Value of the complex: (1,32)
Value of the complex: (0,1)
Value of the complex: (3,9)
Value of the complex: (12,4)
Value of the complex: (3,4)
End of Aldor function f

Now let's use the classes ...
Value of the complex: (8,16)
Value of the complex: (12,24)
Value of the complex: (40,80)
Value of the complex: (30,60)
Value of the complex: (25,50)
End of use of classes

DB pretty complex
Complex name: F
Value of the complex: (231,322)
Complex name: E
Value of the complex: (34,87)
Complex name: D
Value of the complex: (51,62)
Complex name: C
Value of the complex: (12,8)
Complex name: B
Value of the complex: (6,3)
Complex name: A
Value of the complex: (1,2)
End DB pretty complex
```

Here are the steps to be performed to get this result:

1. Make sure that the *Aldor* compiler has the `-Fc++` option (using `axiomxl -hF` and checking that `c++` is an available option) and that `g++` is available
2. Get the files `db.as` and `appli.C` from `example1.tar.gz`
3. Type `axiomxl -Fc++ db.as`
4. Type `axiomxl -Fo db_as.as`
5. Type:

```
g++ -o appli appli.C db_as.o \
    -I$AXIOMXLROOT/include -L$AXIOMXLROOT/lib \
    -laxllib -lfoam -lm
```

6. Run `appli`

12.2 Generic List

This example has been designed to show how parameterized types are handled. We create our own types:

- `ItemCat` is a category for the type of the items of the list.
- `MySI` is a very simple domain which enables to add and output SingleIntegers.
- `MyString` is equivalent to MySI but manipulates Strings of characters.
- `MyList` is a parameterized domain which enables to create lists and compute the addition of all the items. The parameter of this domain is the type of the items.

Here is the actual code:

```
#include "axllib.as"

define ItemCat: Category == with {
     +: (%,%)->%;
     null: %;
     output: % -> ();
}

MySI: ItemCat with {
     bracket: SInt$Machine -> %;
     coerce: % -> SInt$Machine;
} == add {
     Rep ==> SInt$Machine;
     import from Rep, Machine;
     import from SingleInteger;

     bracket(i: SInt$Machine): % == per i;
     coerce(o: %): SInt$Machine == rep(o);

     (o1: %) + (o2: %): % == per (rep(o1) + rep(o2));
     null: % == per coerce(0$SingleInteger);
     output(o: %): () == print << rep(o)::SingleInteger;
}

MyString: ItemCat with {
     bracket: String -> %;
     coerce: % -> String;
} == add {
     Rep ==> String;
     import from Rep;

     bracket(s: String): % == per s;
     coerce(o: %): String == rep(o);

     (o1: %) + (o2: %): % == per (concat(rep(o1),rep(o2)));
     null: % == per "";
     output(o: %): () == print << rep(o);
}

MyList(I: ItemCat): with {
     bracket: () -> %;
```

```
        append:  (I,%) -> %;
        total:   % -> I;
        output:  % -> ();
} == add {
        Rep ==> Union(empty__list: Pointer, real__list: Record(i: I, next: %));
        import from Rep;

        bracket(): % == per [nil];
        append(i: I, o: %): % ==
                if (rep(o) case empty__list)
                then per [[i,per [nil]]];
                else per [[i,o]];
        total(o: %): I == {
                local tmp: % := o;
                local acc: I := null;
                while ( not (rep(tmp) case empty__list)) repeat {
                        acc := acc + rep(tmp).real__list.i;
                        tmp := rep(tmp).real__list.next;
                }
                acc
        }
        output(o: %): () == {
                local tmp: % := o;
                while ( not (rep(tmp) case empty__list)) repeat {
                        print << "[";
                        output(rep(tmp).real__list.i);
                        tmp := rep(tmp).real__list.next;
                        print << "]";
                }
                print << newline;
        }
}

runlist():() == {
        import from String, SInt$Machine, MySI, MyString, MyList(MyString), MyList(MySI),SingleInteger;
        ml__str: MyList(MyString) := append(["Hello,"],append([" I am "],
                                        append(["happy to "],append(["be there."],[]))));
        ml__si : MyList(MySI) := append([coerce(32)],append([coerce(54)],append([coerce(73)],[])));
        s: MyString := total(ml__str);
        i: MySI := total(ml__si);

        print << "List 1: "; output(ml__str); print << newline;
        print << "List 2: "; output(ml__si); print << newline;
        output(s); print << newline;
        output(i); print << newline;
}
```

The function `runlist` is a test function of the above types. In this exercise, we will reproduce the function `runlist` in the `main` function of a C++ program. We will also call `runlist` to check if the results are equivalent.

The following code is thus typed by the user:

```
#include <iostream.h>
#include "mylist_cc.h"

main() {
        MyList<MyString> *ml_str = MyList<MyString>::bracket();
```

```
    ml_str = MyList<MyString>::append(MyString::bracket("be there."),ml_str);
    ml_str = MyList<MyString>::append(MyString::bracket("happy to "),ml_str);
    ml_str = MyList<MyString>::append(MyString::bracket(" I am "),ml_str);
    ml_str = MyList<MyString>::append(MyString::bracket("Hello,"),ml_str);

    MyList<MySI> *ml_si = MyList<MySI>::bracket();
    ml_si = MyList<MySI>::append(MySI::bracket(73),ml_si);
    ml_si = MyList<MySI>::append(MySI::bracket(54),ml_si);
    ml_si = MyList<MySI>::append(MySI::bracket(32),ml_si);

    MyString *s = ml_str->total();
    MySI *i = ml_si->total();

    cout << "----------------------------------------" << endl;
    cout << "   Actual C++ program using Aldor types   " << endl;
    cout << "----------------------------------------" << endl;
    cout << "List 1: "; ml_str->output(); cout << endl;
    cout << "List 2: "; ml_si->output(); cout << endl;
    s->output(); cout << endl;
    i->output(); cout << endl;
    cout << "----------------------------------------" << endl;
    cout << "     Call to the Aldor function runlist     " << endl;
    cout << "----------------------------------------" << endl;
    runlist();
}
```

The different files used in this example are:

- `mylist.as` is the *Aldor* source file of the types described above.
- `appli.C` is the C++ application which makes use of *Aldor* types.

How to compile everything and make it work ? As for the previous example, the steps are the following:

- Make sure that the *Aldor* compiler has the `-Fc++` option (using `axiomxl -hF` and checking that `c++` is an available option) and that `g++` is available
- Get the files `mylist.as` and `appli.C` from `example2.tar.gz`
- Type `axiomxl -Fc++ mylist.as`
- Type `axiomxl -Fo mylist_as.as`
- Type:

```
g++ -o appli appli.C mylist_as.o \
    -I$AXIOMXLROOT/include -L$AXIOMXLROOT/lib -laxllib -lfoam -lm
```

- Run `appli`

The ouput is:

```
----------------------------------------
   Actual C++ program using Aldor types
----------------------------------------
List 1: [Hello,][ I am ][happy to ][be there.]

List 2: [32][54][73]

Hello, I am happy to be there.
159
----------------------------------------
     Call to the Aldor function runlist
```

```
----------------------------------------
List 1: [Hello,][ I am ][happy to ][be there.]

List 2: [32][54][73]

Hello, I am happy to be there.
159
```

Chapter 13

Revision History

Here is an history of the "*Aldor* to C++" tool.

13.1 V1.1 to V1.2

- New option ('-P') to control C++ generation
- Generic handling of basic types correspondence through a file
- Overloading of functions on return type only
- Function pointers

13.2 V1.0 to V1.1

- Handling of parameterized types
- Better type determination
- Operators
- An example from the standard *Aldor* library

Bibliography

[W+94] Stephen M. Watt, Peter A. Broadbery, Samuel S. Dooley, Pietro Iglio, Scott C. Morrison, Jonathan M. Steinbach, Robert S. Sutor, *"AXIOM Library Compiler user Guide,"* The Numerical Algorithms Group Limited, 1994.

[GW97] Marc Gaëtano, Stephen M. Watt, *An Object Model Correspondence for* Aldor *and C++* The FRISCO Consortium, 1997.

[CDWb98] Yannis Chicha, Florence Defaix, Stephen M. Watt, *Automation of the* Aldor*/C++ interface: Technical Reference* The FRISCO Consortium, 1998.

[CDWc98] Yannis Chicha, Florence Defaix, Stephen M. Watt, *A C++to XML translator* The FRISCO Consortium, 1998.